

# Debugging in AnyLogic

Nathaniel Osgood

MIT 15.879

May 16, 2012

# Model Appropriateness Consideration

- Have we built the right model?
- Have we built the model right?

# Have We Built the Right Model?

- This is the province of “validation”
- We can rarely validate the model – only seek to
  - Build confidence
  - Disconfirm it
- This is specific to model purpose
- Here, a lapse is either
  - an oversimplification of the situation
  - An inaccurate “dynamic hypothesis” as to how things work

# Have We Built the Model Right?

- Did we implement our planned model logic as we had intended?
  - Did we want one thing and put in place mechanisms that entailed another thing
  - This is the province of classic testing & quality assurance
    - Peer reviews
    - Testing (e.g. Junit)
- Here, a lapse is typically a model “defect” (*bug*)
  - In this lecture, we will be dealing with identifying this sort of defect

# Debugging: Faults, Failures

- A “fault” is an underlying defect
- A failure is a visible problem, e.g.
  - Model “crashes”
  - Model will not run
  - Model is reporting values that are patently impossible given the implications of our intensions
    - Carcasses arising and walking
    - People recovering form a lifelong illness
    - People moving on a surface that should be impassable (e.g. a river)

# Surprises & Failures

- Often complex models (including ABMs) exhibit surprising emergent properties
  - There may be things we consider very implausible that are jointly implied of various pieces of our model specification
  - There may even be things we consider “impossible” given our intended model structure that are in fact implied by it – we just didn’t realize this!

# Some Model “Surprises” Reflect...

- Mistakes in our implementation (divergence of “what we told the model to do” from “what we intended to tell the model to do”)
  - Typing “ $a/a+b$ ” rather than “ $a/(a+b)$ ”
  - Misunderstanding of how a type of model building block (e.g. a guard in a rate transition) “works”
- Unrealistic aspects of our plan (“what we intended to tell the model to do” had hidden inconsistencies with how the world works)
- Discoveries about what could happen in the world
- We are focusing here on the first of these issues, but need to realize that it often takes time to figure out in which category a given surprise lies!

# What is Debugging?

- Debugging is the process of finding and removing the defects (faults) in our program, based on observations of “failures” or “aberrant behaviour”



# Best Debugging Strategy: Avoiding It!

- Defensive Programming
- Offensive Programming

**We will talk about best practices for these approaches in a separate lecture**

# Offensive Programming: Try to Get Broken Program to Fail Early, Hard

- Asserts: Proactively scan for and flag incorrect assumptions, aborting the program as a result
- Fill memory allocated with illegal values
- Fill object w/illegal data just before deletion
- Set buffers at end of heap, so that overwrites likely trigger page fault
- Setting default values to be illegal in enums
- We will talk about Assertions & Error Handling later this week

# Assertion Goal: Fail Early!

- Alert programmer to misplaced assumptions as early as possible
- Benefits
  - Documents assumptions
  - Reduces likelihood that error will slip through
    - Helps discourage “lazy” handling of only common case
    - Forces developer to deal explicitly with bug before continuing
  - Reduces debugging time
  - Helps improve thoroughness of tests

# Avoid Side Effects in Assertions

- Because assertions may be completely removed from the program, it is unsafe to rely on side effects occurring in them

~~assert ++i < max;~~

Arnold et al. The Java Programming Language, Fourth Edition. 2006.

# Enabling Assertions in Java

- 2 ways
  - Usual: Via java runtime command line
    - enableassertions/-ea[*descriptor*]
    - e.g.
      - enableassertions:com.acme.Plotter
      - enableassertions:com.acme...
    - disableassertions/-da[*descriptor*]
  - Less common: via reflection (ClassLoader)
    - public void **setDefaultAssertionStatus(boolean enabled)**
    - public void **setPackageAssertionStatus(String packageName, boolean enabled)**
    - public void **setClassAssertionStatus(String className, boolean enabled)**

# Enabling Assertions in AnyLogic

The screenshot displays the AnyLogic Advanced [EDUCATIONAL USE ONLY] interface. The main workspace shows a state transition diagram for a 'Person' agent. The diagram includes a 'PregnancyStatus' state with two sub-states: 'NonPregnant' and 'Pregnant'. The 'NonPregnant' state transitions to 'Pregnant' via a 'PerformBirth' event, and 'Pregnant' transitions back to 'NonPregnant' via a 'FinalizeDeath' event. The 'Person' agent has several attributes: 'isInitiallyInfected', 'sex', 'ethnicity', 'InitialAge', 'CurrentAge', and 'FinalizeDeath'. The 'Person' agent also has several methods: 'FertilityRateAgeSexEthnicity', 'PerformBirth', 'EstablishOffspringConnectionsBasedOnMothersConnections', 'EstablishOffspringLocationBasedOnMothersLocation', 'RandomSex', 'RandomEthnicity', 'RandomAge', 'isInReproductiveYears', and 'IsInfected'.

The 'Simulation - Simulation Experiment' properties window is open, showing the 'Advanced' tab. The 'Java Machine Arguments' field is highlighted in red and contains the text '-enableassertions'. Other fields include 'Maximum Available Memory' (1024 Mb) and 'Command-line Arguments'.

Tab	Field	Value
Advanced	Maximum Available Memory	1024 Mb
	Java Machine Arguments	-enableassertions
General	Application options	(will not be applied when model runs as applet)
	Command-line Arguments	
Parameters	Load root object from snapshot	<input type="checkbox"/>
	Browse...	

# Assertions in Later AnyLogic Versions

- In some later AnyLogic versions, should enable assertions only in the model itself
- This is simple to do
  - Uses the package name
- More details on this are available on request

# AspectJ and Eclipse

- AspectJ is a language that allows for succinctly describing “cross cutting” functionality in programs – such as tracing or logging requests
- AspectJ can automatically insert tracing instrumentation into our code
  - This gives us many of the benefits of manual tracing program execution without the need for the markup & mark-down work
- If time permits, we will present this method on Friday



# A Powerful Debugging Approach

- Simplify error occurrence as much as possible
  - Locate fault source
    - Gather data or context that reproduces problem
      - Rip out whole areas of model to see simplest condition that (sometimes just seeing what eliminates error immediately clues in to what it might be)
    - Record what have done
- do
- Analyze data & form hypothesis about defect
  - Determine how to prove/disprove hypothesis
  - Prove or disprove hypothesis
  - Think about defect
- Until can fix defect
- Look for similar errors that may not yet be found
  - Figure out what about *process* left vulnerable to this error

# Important Elements

- “Localizing” problem (Simplifying model & input until discover minimum required mechanism)
  - Save away original model (so don’t modify!)
  - Comparing good & bad versions: What is different?
  - Note down what does & does not work
  - Seeing path of execution (particularly around fault location)
- Alternate between thinking & experimenting
- Observing model state (“situation”) at points preceding error
- Compare with previous versions that were working
- Read error messages given by AnyLogic
- Confirming certain assumptions are true prior to error
- Talk with someone about issue/perform a peer review
- Specify and investigate top hypotheses

# Debugging AnyLogic

- AnyLogic's researcher version now contains a debugger
- You can attach to AnyLogic from debuggers such as Eclipse
  - The key thing is to set anylogic to use a port

# Debugging Options

- Debugging is the process of locating and fixing the faults behind observed failures
- Using output for manual tracing & reporting
  - A valuable option here is to use this interactively
- Using model navigation mechanisms to inspect information about the model
- Using AspectJ for tracing/logging
- Using tools like log4j for customizable logging
- Using an external debugger (e.g. via eclipse)
- Using AnyLogic Professional/Research debugger

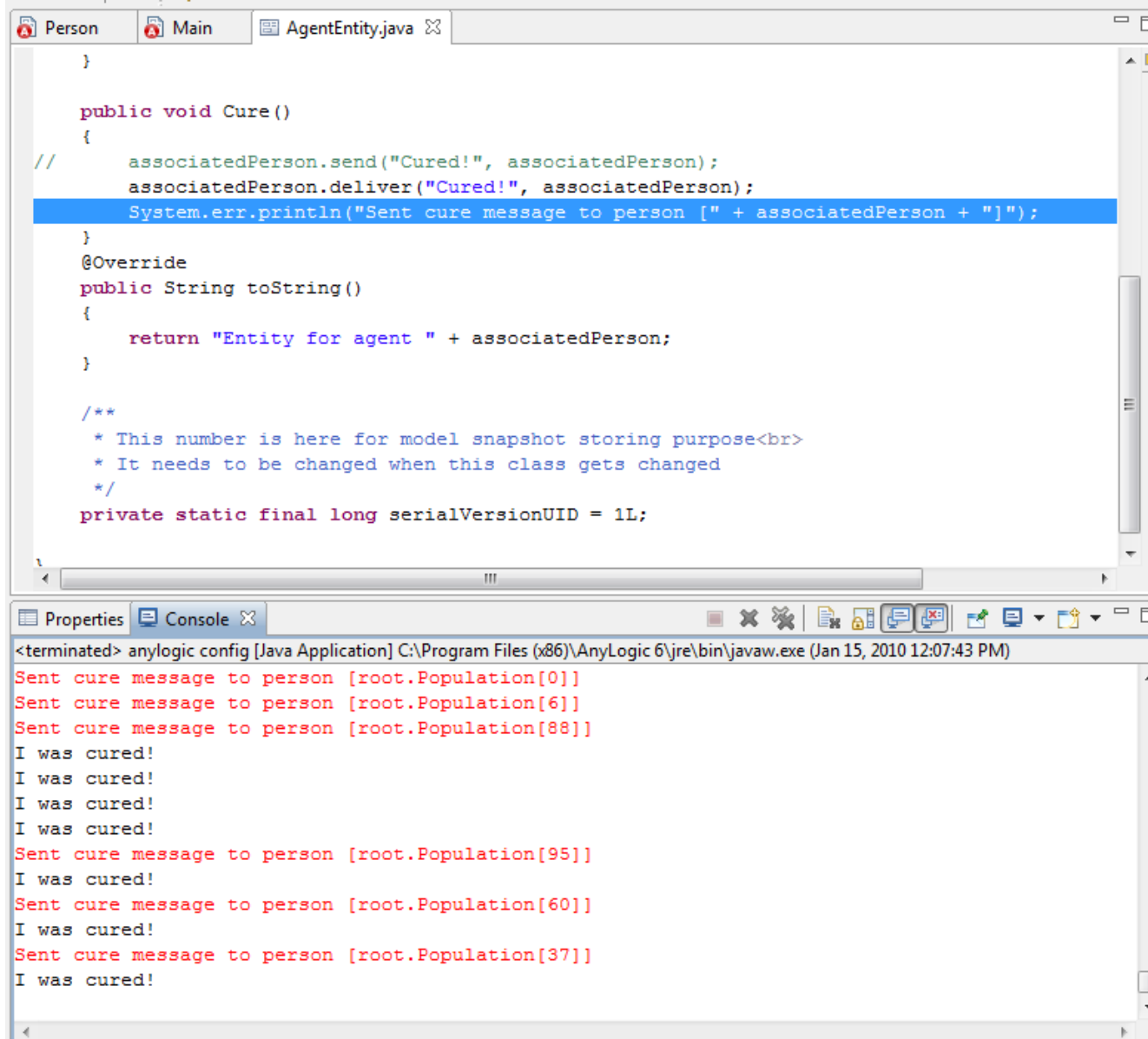
# Using output for manual tracing & reporting

- Pros
  - Minimal learning curve
  - Flexible
  - Easily targeted
- Cons
  - Requires time-consuming manual
    - “markup”
    - de-markup
  - Can require many build/simulation iterations to localize problem
  - Limited capacity of console

# Output to the Console: How To

- `System.err.println(String)`
  - `System.err.println("Sent cure message to person [" + associatedPerson + "]);`
  - This will appear in red
- `println(String)`
- `System.out.println(String)`

# Use in AnyLogic



The screenshot displays an IDE window with the following components:

- Code Editor:** Shows the `AgentEntity.java` file. The `Cure()` method is highlighted in blue. The code includes a `send` call, a `deliver` call, and a `println` statement. Other methods like `toString()` and a `serialVersionUID` are also visible.
- Console:** Shows the output of the application. It includes error messages from `System.err.println` and messages from the `Person` agent, such as "I was cured!".

```
    }  
  
    public void Cure()  
    {  
        // associatedPerson.send("Cured!", associatedPerson);  
        associatedPerson.deliver("Cured!", associatedPerson);  
        System.err.println("Sent cure message to person [" + associatedPerson + "]);  
    }  
    @Override  
    public String toString()  
    {  
        return "Entity for agent " + associatedPerson;  
    }  
  
    /**  
     * This number is here for model snapshot storing purpose<br>  
     * It needs to be changed when this class gets changed  
     */  
    private static final long serialVersionUID = 1L;
```

<terminated> anylogic config [Java Application] C:\Program Files (x86)\AnyLogic 6\jre\bin\javaw.exe (Jan 15, 2010 12:07:43 PM)  
Sent cure message to person [root.Population[0]]  
Sent cure message to person [root.Population[6]]  
Sent cure message to person [root.Population[88]]  
I was cured!  
I was cured!  
I was cured!  
I was cured!  
Sent cure message to person [root.Population[95]]  
I was cured!  
Sent cure message to person [root.Population[60]]  
I was cured!  
Sent cure message to person [root.Population[37]]  
I was cured!

# Interactive reporting

- AnyLogic's support of interactive mechanisms allows us to custom-trigger reporting through user interface actions
  - Button push
  - Mouse click
- We can also use elements like sliders to change things in a way that hints as to the nature of a problem
- This reporting may be
  - Custom-built for debugging
  - Built in, but not typically used here



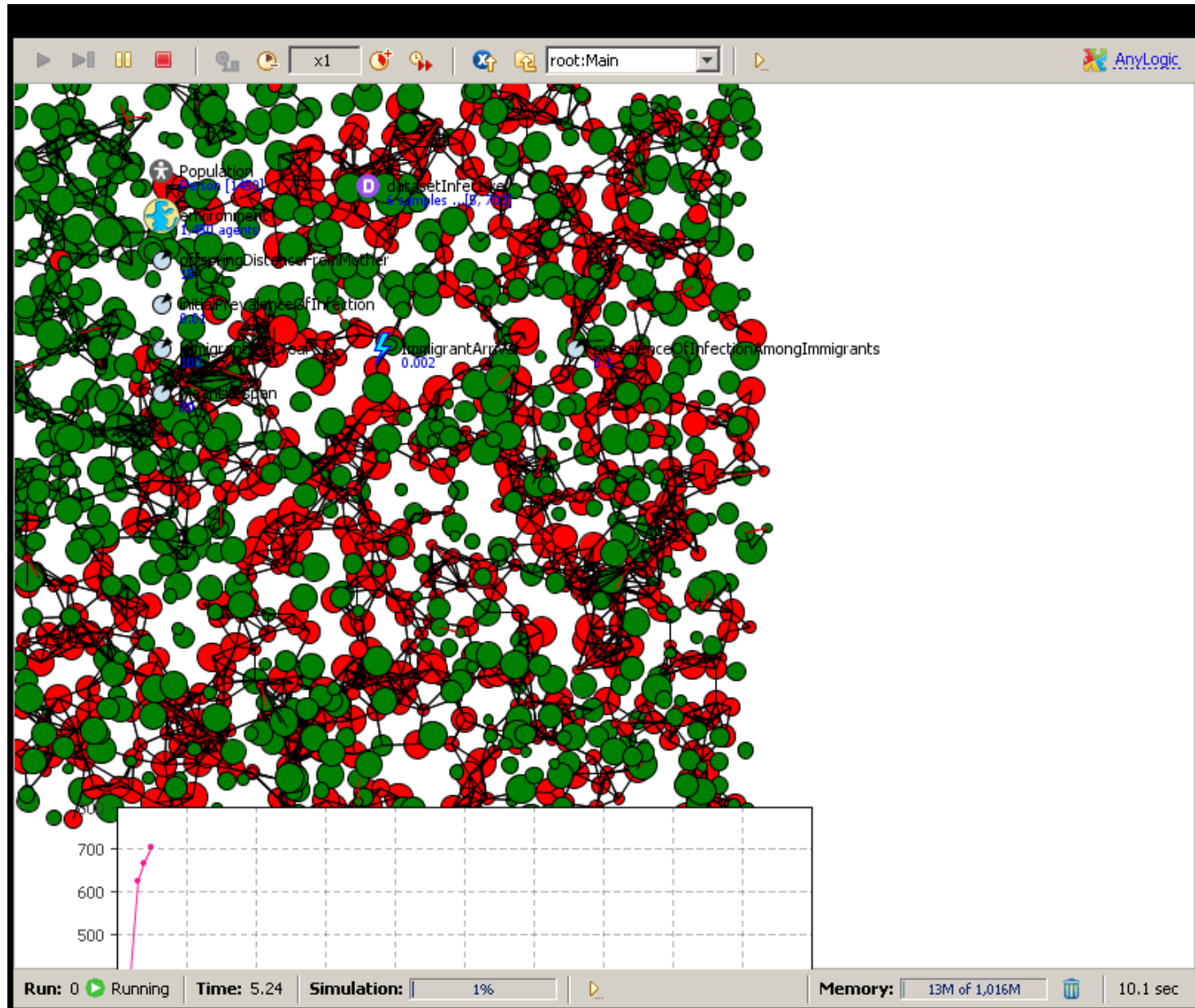


Hands on Model Use Ahead

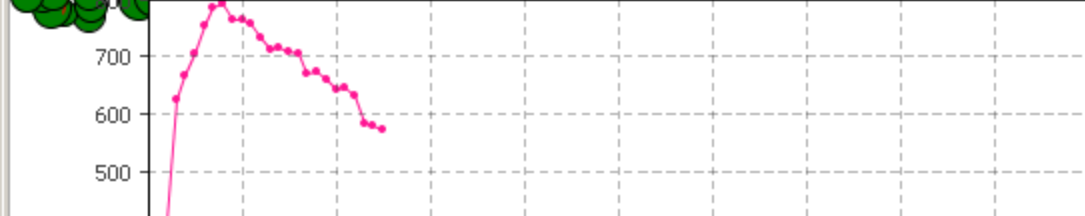
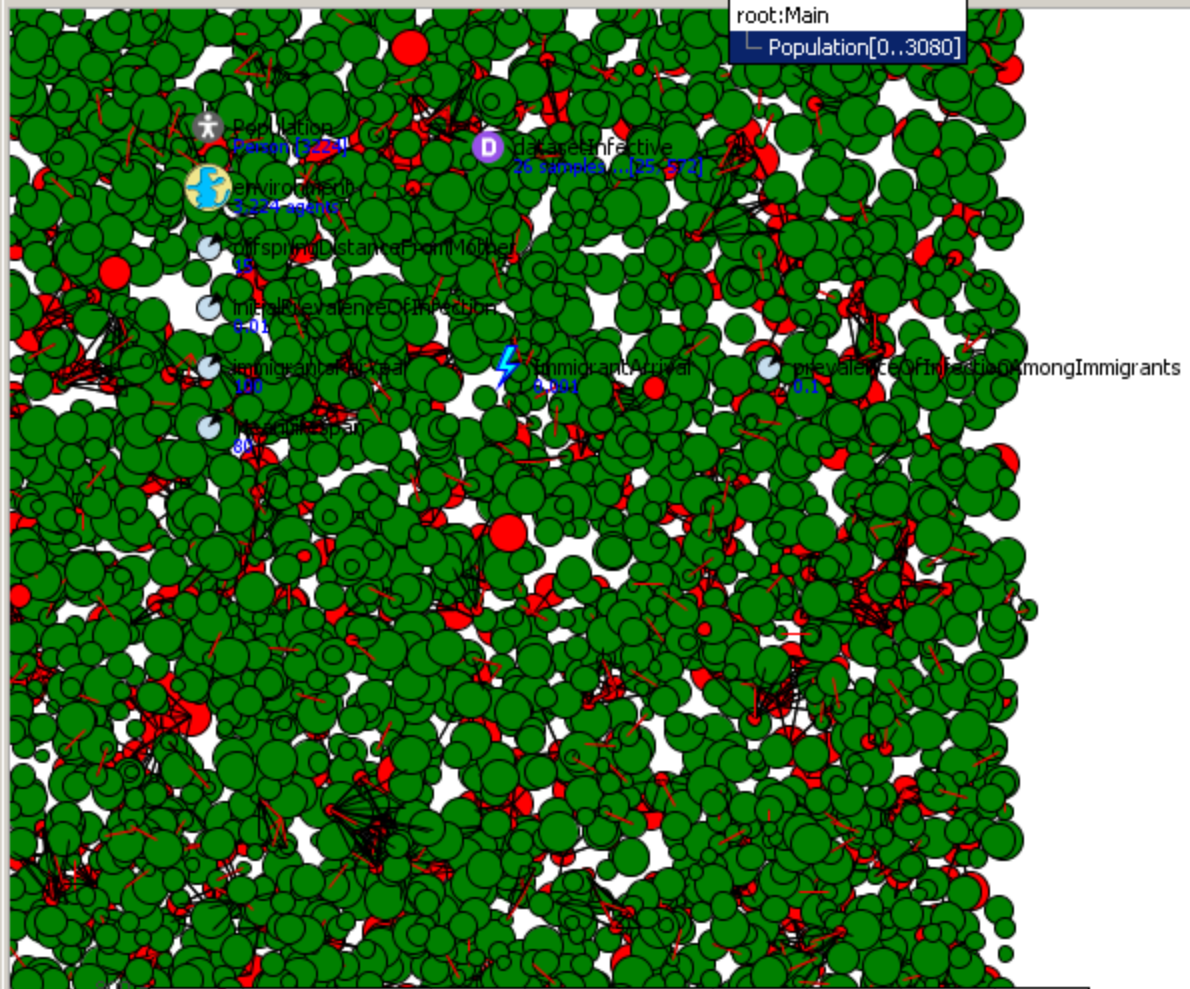
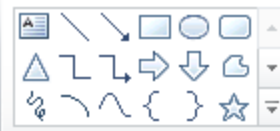


Load Provided Shared Model:  
**ABMModelWithBirthDeath**

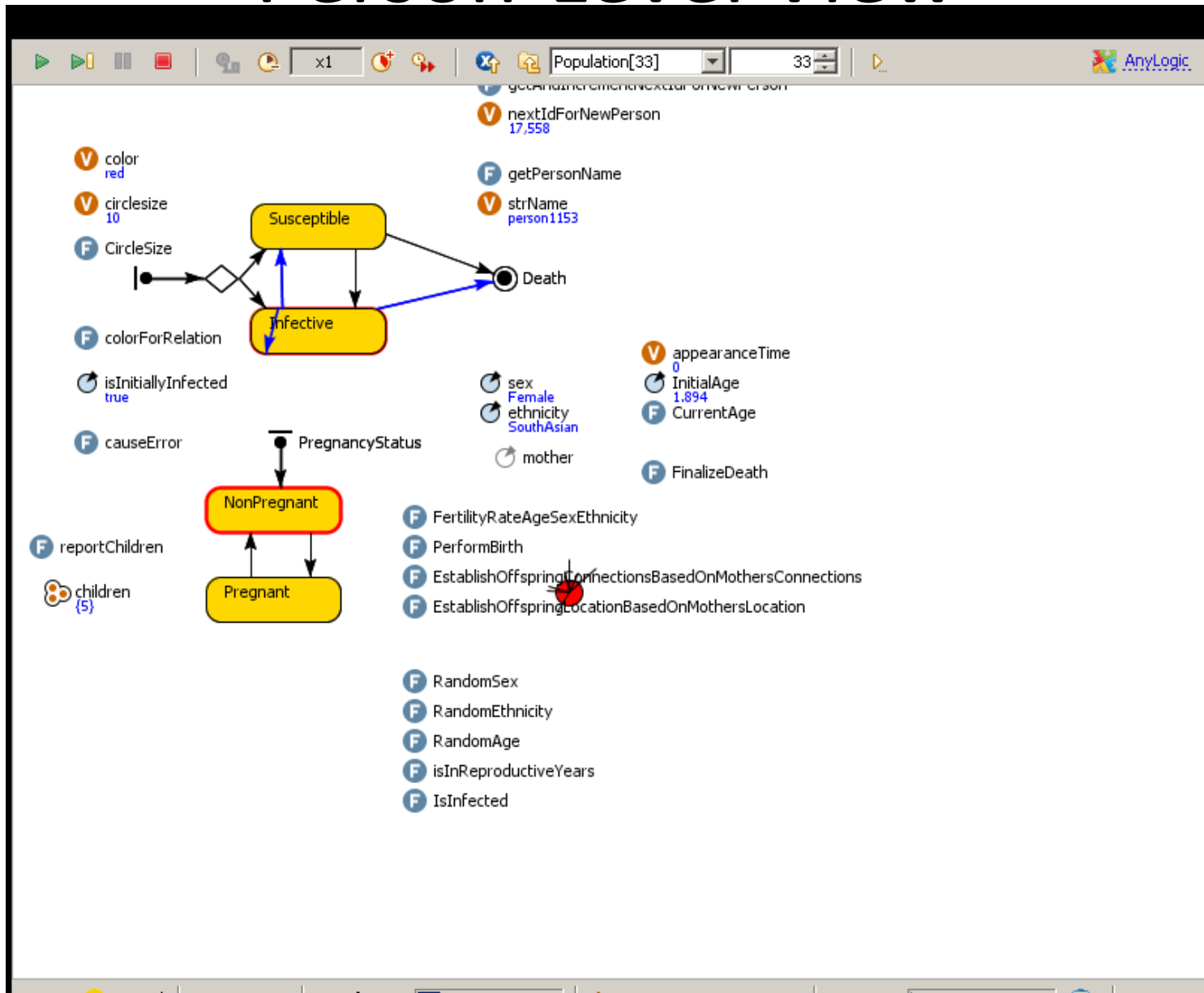
# Population View



root:Main  
Population[0..3080]



# Person-Level View



# Examining Contents of Collection

The screenshot displays the AnyLogic simulation environment for a model named 'ABMModel2'. The main workspace shows a state transition diagram with the following states and transitions:

- Susceptible** (yellow rounded rectangle) transitions to **Infective** (yellow rounded rectangle) via a blue arrow.
- Infective** transitions to **Death** (black circle with a white center) via a blue arrow.
- NonPregnant** (red rounded rectangle) transitions to **Pregnant** (yellow rounded rectangle) via a black arrow.
- Pregnant** transitions to **NonPregnant** via a black arrow.
- NonPregnant** transitions to **Death** via a black arrow.
- Pregnant** transitions to **Death** via a black arrow.
- NonPregnant** transitions to **NonPregnant** via a black arrow.
- Pregnant** transitions to **Pregnant** via a black arrow.

Variables and functions are listed on the right side of the workspace:

- color**: red
- circlesize**: 10
- CircleSize**: (function)
- colorForRelation**: (function)
- isInitiallyInfected**: true
- causeError**: (function)
- nextIdForNewPerson**: 17,558
- getPersonName**: (function)
- strName**: person1153
- appearanceTime**: 0
- InitialAge**: 1,894
- CurrentAge**: (function)
- FinalizeDeath**: (function)
- sex**: Female
- ethnicity**: SouthAsian
- mother**: (function)
- FertilityRateAgeSexEthnicity**: (function)
- PerformBirth**: (function)
- EstablishOffspringConnectionsBasedOnMothersConnections**: (function)
- EstablishOffspringLocationBasedOnMothersLocation**: (function)
- RandomSex**: (function)
- RandomEthnicity**: (function)
- RandomAge**: (function)
- isInReproductiveYears**: (function)
- IsInfected**: (function)

A collection window titled **children** is open, showing 5 elements:

```
children
5 elements
0: root.Population[907]
1: root.Population[-1]
2: root.Population[2339]
3: root.Population[3252]
4: root.Population[3437]
```

The bottom status bar shows: Run: 0 Paused, Time: 63.75, Simulation: 16%, Memory: 20M of 1,016M, 79.3 sec.

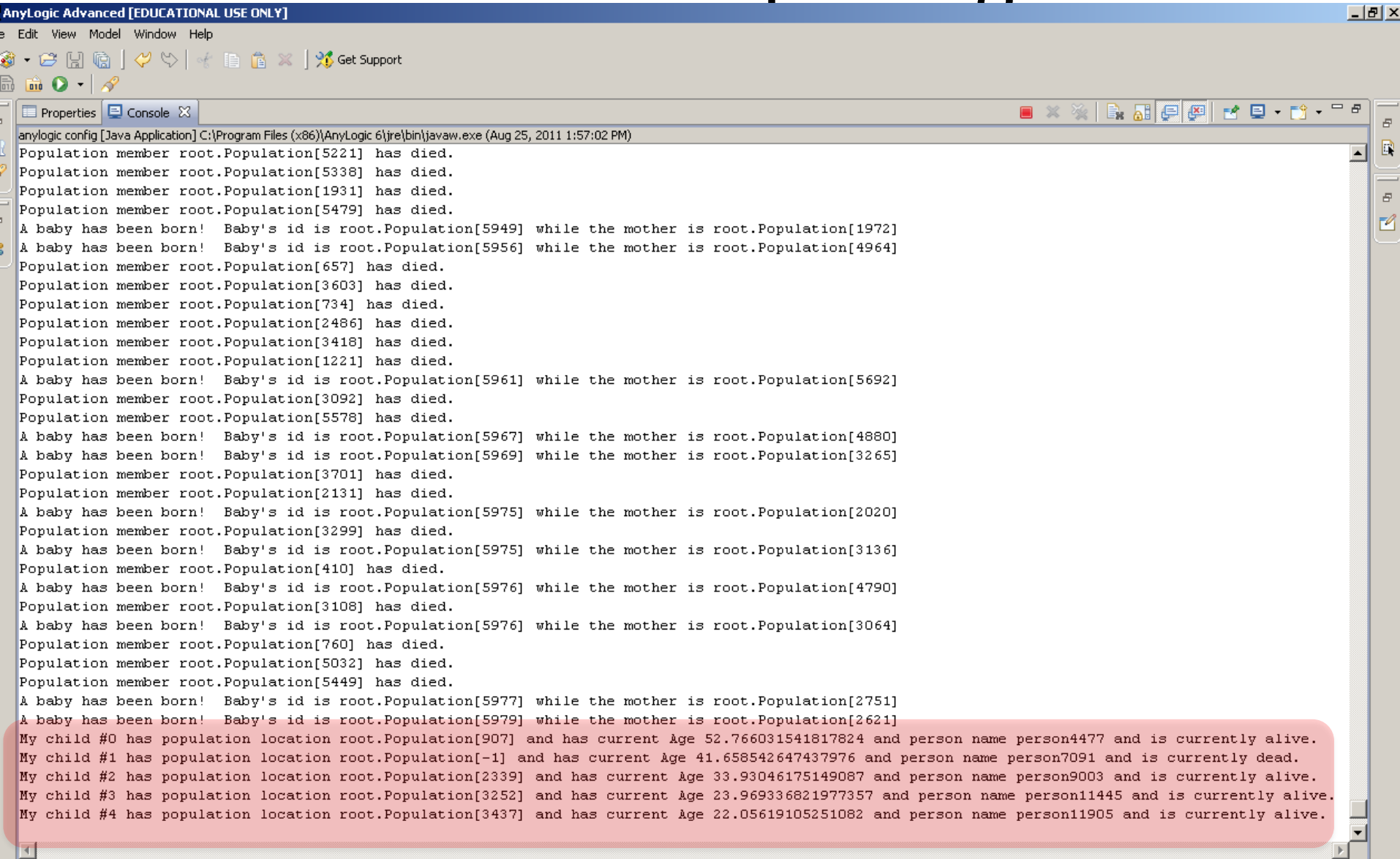
# Examining Contents of Collection

The screenshot displays the AnyLogic software interface for a simulation model. The main workspace shows a state transition diagram with states: Susceptible, Infective, NonPregnant, and Pregnant. A red circle highlights the 'EstablishOffspringLocationBasedOnMothersLocation' function, with a red arrow pointing to it. A yellow box highlights the 'children' collection, which contains 5 elements:

- 0: root.Population[907]
- 1: root.Population[-1]
- 2: root.Population[2339]
- 3: root.Population[3252]
- 4: root.Population[3437]

Other visible elements include a toolbar at the top, a status bar at the bottom showing 'Run: 0 Paused', 'Time: 63.75', 'Simulation: 16%', and 'Memory: 20M of 1,016M'. A large red text overlay at the bottom right reads 'Pause model execution Click here'.

# Custom Reporting



The screenshot shows the AnyLogic Advanced software interface. The title bar reads "AnyLogic Advanced [EDUCATIONAL USE ONLY]". The menu bar includes "File", "Edit", "View", "Model", "Window", and "Help". The toolbar contains various icons for file operations and simulation control. The main window is titled "Console" and displays the following text:

```
anylogic config [Java Application] C:\Program Files (x86)\AnyLogic 6\jre\bin\javaw.exe (Aug 25, 2011 1:57:02 PM)
Population member root.Population[5221] has died.
Population member root.Population[5338] has died.
Population member root.Population[1931] has died.
Population member root.Population[5479] has died.
A baby has been born! Baby's id is root.Population[5949] while the mother is root.Population[1972]
A baby has been born! Baby's id is root.Population[5956] while the mother is root.Population[4964]
Population member root.Population[657] has died.
Population member root.Population[3603] has died.
Population member root.Population[734] has died.
Population member root.Population[2486] has died.
Population member root.Population[3418] has died.
Population member root.Population[1221] has died.
A baby has been born! Baby's id is root.Population[5961] while the mother is root.Population[5692]
Population member root.Population[3092] has died.
Population member root.Population[5578] has died.
A baby has been born! Baby's id is root.Population[5967] while the mother is root.Population[4880]
A baby has been born! Baby's id is root.Population[5969] while the mother is root.Population[3265]
Population member root.Population[3701] has died.
Population member root.Population[2131] has died.
A baby has been born! Baby's id is root.Population[5975] while the mother is root.Population[2020]
Population member root.Population[3299] has died.
A baby has been born! Baby's id is root.Population[5975] while the mother is root.Population[3136]
Population member root.Population[410] has died.
A baby has been born! Baby's id is root.Population[5976] while the mother is root.Population[4790]
Population member root.Population[3108] has died.
A baby has been born! Baby's id is root.Population[5976] while the mother is root.Population[3064]
Population member root.Population[760] has died.
Population member root.Population[5032] has died.
Population member root.Population[5449] has died.
A baby has been born! Baby's id is root.Population[5977] while the mother is root.Population[2751]
A baby has been born! Baby's id is root.Population[5979] while the mother is root.Population[2621]
My child #0 has population location root.Population[907] and has current Age 52.766031541817824 and person name person4477 and is currently alive.
My child #1 has population location root.Population[-1] and has current Age 41.658542647437976 and person name person7091 and is currently dead.
My child #2 has population location root.Population[2339] and has current Age 33.93046175149087 and person name person9003 and is currently alive.
My child #3 has population location root.Population[3252] and has current Age 23.969336821977357 and person name person11445 and is currently alive.
My child #4 has population location root.Population[3437] and has current Age 22.05619105251082 and person name person11905 and is currently alive.
```

# Logging

- *Logging* is the process of recording a record (trace) of events during program execution
  - *Recording can be made to a database, files, text console, etc.*
- Logging can be performed at a variety of levels of detail
- Log4j is one logging framework



# Logging with Log4j

- Use of config files to configure
- Different levels of logger
  - TRACE, DEBUG, INFO, WARN, ERROR and FATAL
- A given logger can be associated with Multiple output streams
- Doing error uploads to a server
- Sending email (?)

```
public class Logger {  
  
    // Creation & retrieval methods:  
    public static Logger getRootLogger();  
    public static Logger getLogger(String name);  
  
    // printing methods:  
    public void trace(Object message);  
    public void debug(Object message);  
    public void info(Object message);  
    public void warn(Object message);  
    public void error(Object message);  
    public void fatal(Object message);  
  
    // generic printing method:  
    public void log(Level l, Object message);  
}
```

# Example use of Log4j

```
// get a logger instance named "com.foo"  
Logger logger = Logger.getLogger("com.foo");  
  
logger.warn("Low fuel level.");  
  
    logger.info("general information");  
    // This request is disabled, because DEBUG < INFO.  
logger.debug("Starting search for nearest gas  
station.");
```

# Config File

Here are example configuration files

**# Set root logger level to DEBUG and its only appender to A1.**

**log4j.rootLogger=DEBUG, A1**

**# A1 is set to be a ConsoleAppender.**

**log4j.appender.A1=org.apache.log4j.ConsoleAppender**

**# A1 uses PatternLayout.**

**log4j.appender.A1.layout=org.apache.log4j.PatternLayout**

**log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p  
%c %x - %m%n**

# Config File: Suppressing Selective Information

```
log4j.rootLogger=DEBUG, A1
```

```
log4j.appender.A1=org.apache.log4j.ConsoleAppender
```

```
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
```

```
# Print the date in ISO 8601 format
```

```
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p  
%c - %m%n
```

```
# Print only messages of level WARN or above in the  
package com.foo.
```

```
log4j.logger.com.foo=WARN
```

# Multiple Outputs

- `log4j.rootLogger=debug, stdout, R`  
`log4j.appender.stdout=org.apache.log4j.ConsoleAppender`  
`log4j.appender.stdout.layout=org.apache.log4j.PatternLayout`
- # Pattern to output the caller's file name and line number.  
`log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) -  
%m%n`
- `log4j.appender.R=org.apache.log4j.RollingFileAppender`  
`log4j.appender.R.File=example.log`  
`log4j.appender.R.MaxFileSize=100KB`
- # Keep one backup file `log4j.appender.R.MaxBackupIndex=1`  
`log4j.appender.R.layout=org.apache.log4j.PatternLayout`  
`log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n`

# Using the External Eclipse Debugger with AnyLogic

# External Debugging in Eclipse

- The “Eclipse” editor is one of the most popular extant software development tools
- Eclipse offers plug-ins of many sorts
  - Debuggers
  - Profilers
  - Visualization tools
  - Version control of models
- Eclipse can be used to debug AnyLogic models at the Java source-code level



# Overview: Setting up External Eclipse Debugging in AnyLogic

- In anylogic, Set the jvm options for socket based debugging (e.g. eclipse)
  - go to "Properties" on the "Simulation" to run for the anylogic model
  - Set the "Java Machine Arguments" as follows:  
`-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8321`
- in eclipse, create a debug configuration
  - use "Remote Java Application"
    - no project
    - for "Connection Type", select "Standard (Socket Attach)"
    - for "Connection properties", Use
      - Host: localhost
      - Port 8321

# Steps Required for Eclipse Debugging

- One time set-up for a particular model
  - Set up AnyLogic to allow debugging connections
  - Set up Eclipse to know
    - How to connect to AnyLogic
    - Where to look for source code files
- Every time want to debug
  - Go to Eclipse
  - Tell debugger to connect to AnyLogic process
  - Interrupt process
  - Set breakpoints, etc.

# One-Time Setup In AnyLogic

- `-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8321`
- These go under the "Advanced" tab of the simulation run to use

# Setting up Debug Configurations

The screenshot displays the Eclipse IDE interface with the 'Debug' menu open. The menu options include:

- Resume
- Suspend
- Terminate
- Step Into
- Step Over
- Step Return
- Run to Line
- Use Step Filters (Shift+F5)
- Run (Ctrl+F11)
- Debug (F11)
- Profile
- Profile History
- Profile As
- Profile Configurations...
- Run History
- Run As
- Run Configurations...
- Debug History
- Debug As
- Debug Configurations... (highlighted)
- Toggle Breakpoint (Ctrl+Shift+B)
- Toggle Line Breakpoint
- Toggle Method Breakpoint
- Toggle Watchpoint
- Skip All Breakpoints
- Remove All Breakpoints
- Add Java Exception Breakpoint...
- Add Class Load Breakpoint...
- All References...
- All Instances... (Ctrl+Shift+N)
- Watch
- Inspect (Ctrl+Shift+I)
- Display (Ctrl+Shift+D)
- Execute (Ctrl+U)
- Force Return (Alt+Shift+F)
- Step Into Selection
- External Tools

The IDE shows the following components:

- Editor:** Displays the source code for `VensimSimulationController` with a breakpoint at line 5. The code includes fields like `logger`, `branchNumber`, `counter`, `curTime`, `decisionTreeStartTime`, and `modelFileName`. A console output shows `localhost:8321]`.
- Outline:** Shows the project structure for `ca.usask.cs.Gui`, including `UseExistingTreeGui` and various listeners and UI components.
- Bottom Panel:** Shows the package explorer with files like `Main.java`, `LoadXML.java`, `InterfaceHandler.java`, and `CreateNewTreeGui.java`.

The status bar at the bottom indicates 'Writable', 'Smart Insert', and '209 : 9'.

# Set up: Creating a Debugging Configuration in Eclipse

Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

Name: Anylogic Application

Connect Source Common

Project:  Browse...

Connection Type: Standard (Socket Attach)

Connection Properties:

Host: localhost

Port: 8321

Allow termination of remote VM

- Eclipse Application
  - edu.usask.cs.silverRCP.product
- Java Applet
- Java Application
  - HTMLinksToFiles
  - Main
  - Run main class
  - TestJavaDecisionTree4
- JUnit
- JUnit Plug-in Test
- OSGi Framework
- Remote Java Application
  - Anylogic Application**
- Task Context Plug-in Test
- Task Context Test

# Setting Up Source Code Folders

The screenshot shows the Eclipse IDE's 'Debug Configurations' dialog box. The title bar reads 'Debug Configurations' and the subtitle is 'Create, manage, and run configurations'. Below the subtitle, it says 'Attach to a Java virtual machine accepting debug connections'. On the right side of the title bar, there is a green bug icon.

The main area is divided into two panes. The left pane is a tree view showing various configuration types. Under 'Remote Java Application', the 'Anylogic Application' configuration is selected and highlighted. The right pane is titled 'Name: Anylogic Application' and has three tabs: 'Connect', 'Source', and 'Common'. The 'Source' tab is active, showing a list of 'Source Lookup Path' entries:

- src.generated - C:\Users\Nate\AnyLogicWorkspace\EclipseDebuggingExample\_BUILD
- anqiV2 - C:\Users\osgood\AnyLogicWorkspace\AnqiV2\_BUILD\classes
- anqiV2 - C:\Users\osgood\AnyLogicWorkspace\AnqiV2\_BUILD\src.generated
- Default

To the right of this list are several control buttons: 'Add...', 'Edit...', 'Remove', 'Up', 'Down', and 'Restore Default'.

# Add Source Folder

Debug - WenyiDecisionTree12\_2010/trunk/treeinterfacev3/src/ca/usask/cs/Gui/UseExistingTreeGui.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

Name: Anylogic Application

Connect Source Common

Add File System Directory

File system folder

Specify folder and whether subfolders should be searched

Directory:

Search subfolders

Directory Selection

Choose directory to add:

- Nate
  - .alice
  - .AnyLogicProfessional**
  - .AnyLogicUniversity [Date created: 10/31/2010 8:27 PM]
  - Config
  - Workspace
    - .metadata
    - EclipseDebuggingExample
    - EclipseDebuggingExample\_BUILD
    - .settings
    - classes
    - src.generated
    - abrepedelwithbiathdeath

Folder: src.generated

Make New Folder OK Cancel

Variables Breakpoints

RuntimeException

NullPointerException: caught and un

RuntimeException: caught and un

BuildPictureGui\$dotSaveListener [

BuildPictureGui\$dotSaveListener [

CreateNewTreeGui\$CreateTreeLis

CreateNewTreeGui\$CreateTreeLis

CreateScenarioDialog [line: 436] -

HTMLinksToFiles [line: 27] - main

VensimSimulationCont UseExistr

Console

11:56 AM 5/16/2012

**The AnyLogic Workspace is Located under the User Folder**

# Once Set up, Can...

- Set breakpoints
- See the variables, with symbolic information
- Suggestions
  - Set a breakpoint on a thrown runtime exception (regardless of whether caught)
  - Throw a caught runtime exception from model startup code
  - When catch this in Eclipse, can then use to set breakpoints (including in other files)



# Start AnyLogic Model (Experiment with Extra Debugging JVM Arguments)

The screenshot displays the AnyLogic Advanced software interface, showing a simulation experiment setup for a debugging session. The main window is titled "DebuggingSession - Simulation Experiment".

**Project Explorer (Left):** Shows the project structure for "EclipseDebuggingExample". The "Main" package is expanded, and the "DebuggingSession: Main" file is selected. A context menu is open over this file, showing options like "New", "Open with", "Open...", "Cut", "Copy", "Paste", "Delete", "Refresh", and "Run".

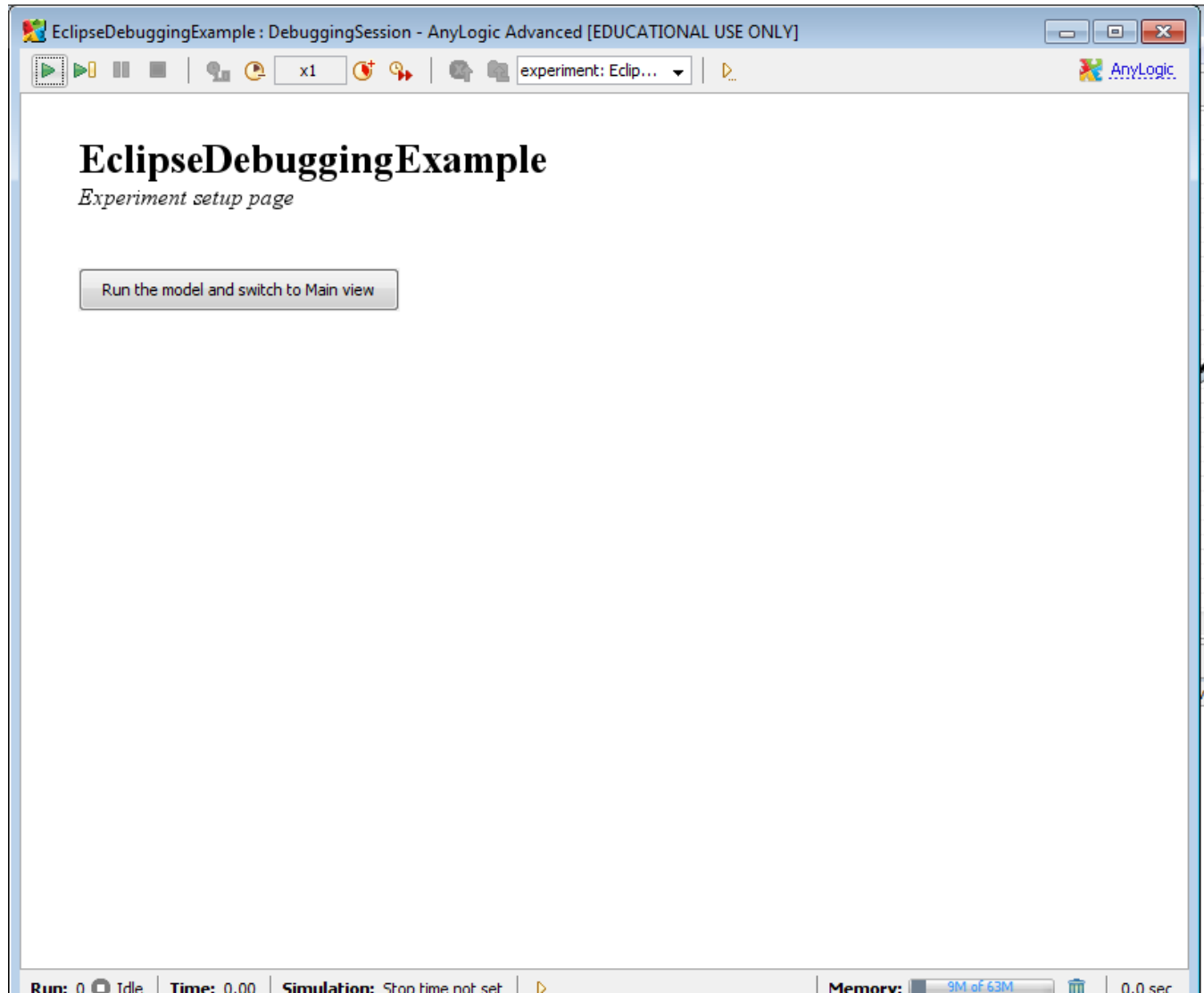
**Diagram (Center):** A diagram showing the model structure. The "Main" package contains several objects: "populationSize", "Population [..]", "environment", "offspringDistanceFromMother", "initialPrevalenceOfInfection", "immigrantsPerYear", "MeanLifespan", "datasetInfective", "ImmigrantArrival", "prevalenceOfInfectionAmongImmigrants", and "TriggerDebugger".

**Properties Panel (Bottom):** Shows the configuration for the "DebuggingSession - Simulation Experiment".

- General:** Name: DebuggingSession; Main active object class (root): Main; Ignore:
- Advanced:** Random number generation:  Fixed seed (reproducible simulation runs); Seed Value: 1
- Parameters:** offspringDistanceFromMother: 15 /\* half a distance outside of perimeter \*/; initialPrevalenceOfInfection: 0.01; immigrantsPerYear: 100; prevalenceOfInfectionAmongImmigrants: 0.10; MeanLifespan: 80.0

**Right Panel:** Shows a list of model components and actions, including "Model", "Parameter", "Flow Aux...", "Stock Vari...", "Event", "Dynamic...", "Plain Vari...", "Collectio...", "Function", "Table Fun...", "Port", "Connector", "Entry Point", "State", "Transition", "Initial Stat...", "Branch", "History St...", "Final State", "Environm...", "Action", "Analysis", "Presentati...", "Connectivi...", and "Enterprise...".

# Leave on Opening Screen for Now (So We can Set up Eclipse)



# In Eclipse, Open “Debug” Perspective

The screenshot displays the Eclipse IDE in the 'Debug' perspective. The top toolbar shows the 'Debug' button (a green bug icon) highlighted, with a context menu open over it. The menu options include 'Aspect Visualization', 'Debug', 'Java Browsing', and 'Other...'. The central editor shows the source code of 'Person.java', with the 'PerformBirth()' method selected. The code includes comments and method calls like 'EstablishOffspringConnectionsBasedOnMothersConnection'. The left Project Explorer shows a project named 'AnyLogicDebugProject' with a package structure including 'src', 'MethodCalls.aj', and 'AnyLogicTracing3'. The right sidebar contains the 'Problems' view, which lists 841 errors, 281 warnings, and 0 others. The bottom status bar shows 'Writable', 'Smart Insert', '520 : 1', and 'Build Project'.

```
void PerformBirth() {
    Person mother = this;
    Person offspring = get_Main().add_Population(
        traceIn("A baby has been born! Baby's id is " +
            // establish connections of infant
            EstablishOffspringConnectionsBasedOnMothersConnection
            // now position the baby to be close to the mother
            EstablishOffspringLocationBasedOnMothersLocation(
                ..
            )
        ..
    )
}

void EstablishOffspringConnectionsBasedOnMothersConnection() {
    // now establish links between the baby and all connections
}

if (mother.getConnections() != null) // guard against null
    for (Agent a : mother.getConnections())
        >> {
            >> Person p = (Person) a;
            >> offspring.connectTo(p);
            >> }
}

// Finally, establish a link between the baby and the mother
// (we do this last so we don't have to worry about the mother's connections
// the mother's connections is to this offspring
offspring.connectTo(this);
```

# Start Debugger

The screenshot shows the Eclipse IDE interface with the following components:

- Top Bar:** Eclipse title bar and menu bar (File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help).
- Left Panel:** Hierarchy view showing the project structure, including `VensimSimulationController` and `Object`.
- Center Panel:** Source code editor showing the `VensimSimulationController` class with variables like `logger`, `branchNumber`, `counter`, `curTime`, `decisionTreeStartTime`, and `modelFileName`.
- Right Panel:** Outline view showing the class structure of `ca.usask.cs.Gui`, including `UseExistingTreeGui` and its methods like `contentPanelLayout()`, `initComponents()`, and `setDefaultValue()`.
- Bottom Panel:** Variables view showing the current state of variables, including `RuntimeException`, `NullPointerException`, and `BuildPictureGui$dotSaveListener`.
- Bottom Right Panel:** Console view showing the output of the program, including the text `st:8321`.

# Following Connection

The screenshot displays the Eclipse IDE interface during a debugging session. The title bar indicates the current file is `UseExistingTreeGui.java` in the package `ca.usask.cs.Gui`.

**Hierarchy View:** Shows the object structure. The root is `VensimSimulationController`, which contains an `Object` containing a `VensimSimulationController 48`. The controller has several attributes: `logger`, `branchNumber`, `counter`, `curTime`, `decisionTreeStartTime`, and `modelFileName`.

**Debug View:** Shows the thread hierarchy for the `AnyLogic Application [Remote Java Application]`. The root is `Java HotSpot(TM) 64-Bit Server VM[localhost:8321]`, which contains several threads: `Daemon Thread [AnyLogic presentation frame manager] (Running)`, `Thread [DestroyJavaVM] (Running)`, `Thread [AnyLogic simulation performance monitor] (Running)`, `Thread [AWT-EventQueue-0] (Running)`, `Daemon Thread [AWT-Windows] (Running)`, and `Thread [AWT-Shutdown] (Running)`.

**Outline View:** Shows the class structure for `ca.usask.cs.Gui`. The class `UseExistingTreeGui 90` is expanded, showing its attributes: `BackListener`, `BuildPictureListener`, `FileChooserListener`, `serialVersionUID : long`, `backBotton : JButton`, `browseFile : JButton`, `interfaceHandler : InterfaceHandler`, `loadVensimUI : LoadVensimGui`, `runTreeBotton : JButton`, `specifyTree : JLabel`, `vensimName : String`, `xmlErrorMsg : JLabel`, `xmlLocaton : JTextField`, `UseExistingTreeGui(LoadVensimGui, InterfaceHa`, `contentPaneLayout() : void`, `initComponents() : void`, and `setDefaultValue() : void`.

**Variables View:** Shows the current state of variables. The `RuntimeException` variable is checked, and its value is `NullPointerException: caught and uncaught`. Other variables include `BuildPictureGui$dotSaveListener [line: 207] - actionPerformed(ActionEvent)`, `BuildPictureGui$dotSaveListener [line: 264] - checkXML()`, `CreateNewTreeGui$CreateTreeListener [line: 1054] - actionPerformed(ActionEvent)`, `CreateNewTreeGui$CreateTreeListener [line: 1055] - actionPerformed(ActionEvent)`, `CreateScenarioDialog [line: 436] - new Anonymous`, and `HTMLLinksToFiles [line: 27] - main(String[])`.

**Console View:** Shows the output of the application, including the `main` method of `HTMLLinksToFiles`.

# Open Up Java Files from the Workspace Folder for this Project to Inspect Source & Set Breakpoints

The screenshot shows the Eclipse IDE interface. An "Open File" dialog box is open, displaying a list of files in the project folder "abmmmodelwithbirthdeath". The file "Person.java" is selected. The dialog box has a search bar and a file name field containing "Person.java". The background shows the Eclipse IDE with a console window at the bottom and a code editor on the right.

Name	Date modified	Type	Size
DebuggingSession.java	5/16/2012 12:30 PM	JAVA File	9 KB
HyperArrays.java	5/16/2012 12:30 PM	JAVA File	1 KB
Main.java	5/16/2012 12:30 PM	JAVA File	24 KB
Person.java	5/16/2012 12:30 PM	JAVA File	29 KB
ProfilingSimulation.java	5/16/2012 12:30 PM	JAVA File	9 KB
Simulation.java	5/16/2012 12:30 PM	JAVA File	9 KB

```
on : ShapeGroup
fectionStatechart : Statechart
initialAge : double
itiationOfPregnancy : TransitionRate
InitiallyInfected : boolean
e : ReplicatedShape<ShapeLine>
val : ShapeOval
regnancyStatus : Statechart
resentation : ShapeTopLevelPresentation
ex : Sex
ansition : TransitionMessage
ansition1 : TransitionRate
ansition2 : TransitionRate
ansition3 : TransitionRate
ansition6 : TransitionRate
..}
new ReplicatedShape<ShapeLine>()
new ShapeOval() {...}
erson(Engine, ActiveObject, ActiveObject
ethnicity_DefaultValue_xjal() : Ethnicity
initialAge_DefaultValue_xjal() : double
sInitiallyInfected_DefaultValue_xjal() : b
ine_createShapeWithStaticProperties(int
ine_Replication() : int
ine_SetDynamicParams_xjal(ShapeLine,
oval_SetDynamicParams_xjal(ShapeOval
sex_DefaultValue_xjal() : Sex
ricleSize() : double
eate() : void
new ShapeTopLevelPresentationGrou
urrentAge() : double
awModelElements(Panel, Graphics2D, bc
awModelElements_Functions_xjal(Panel,
awModelElements_Parameters_xjal(Pan
awModelElements_PlainVariables_xjal(Pa
drawModelElements_Statecharts_xjal(Pan
enterState(short, boolean) : void
```

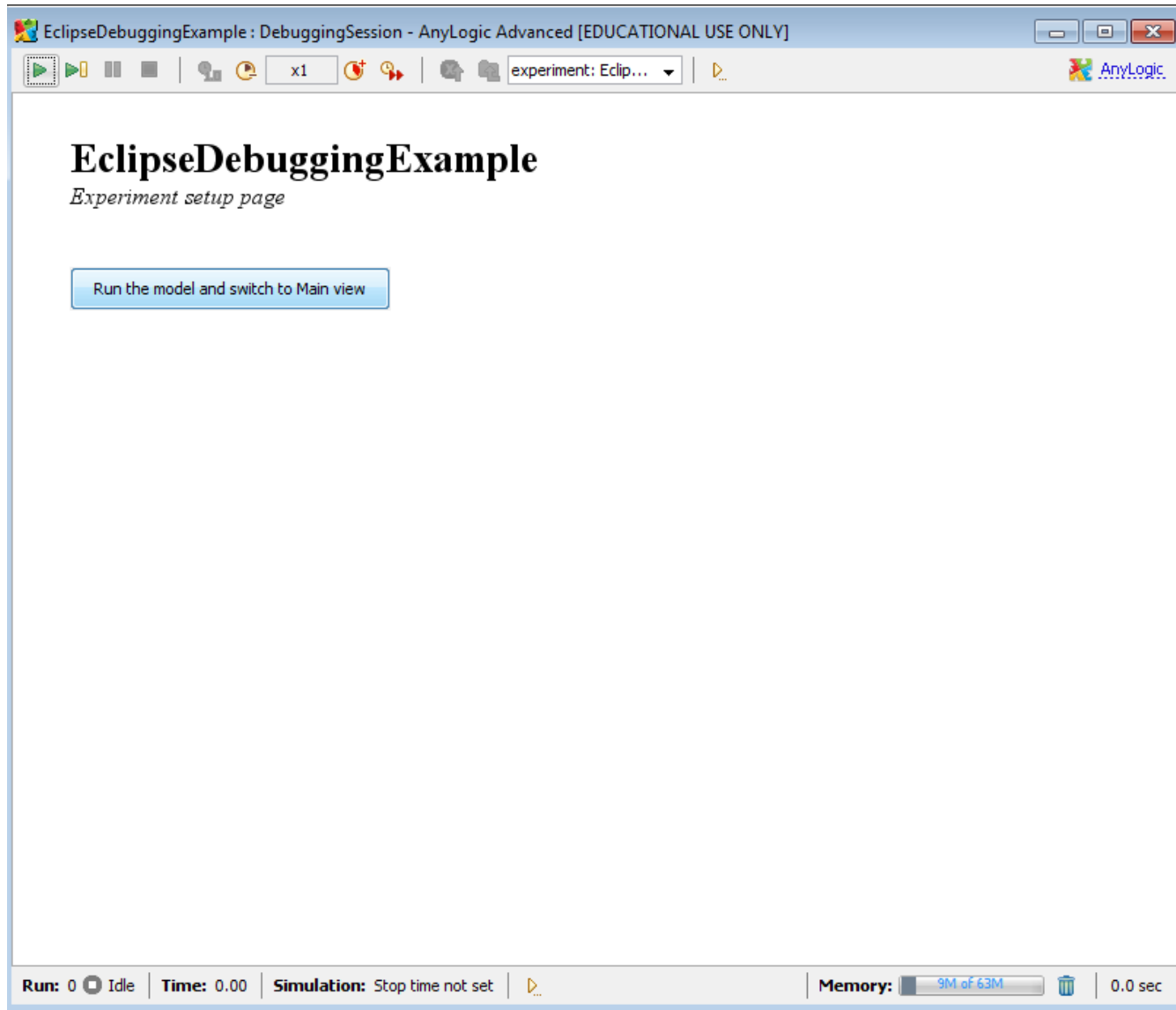
# Now Can Set Breakpoints in Main.java or Elsewhere (Here: Person.java)

The screenshot shows the Eclipse IDE interface. The top toolbar includes icons for File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The main workspace is divided into several panes:

- Debug Console:** Shows the execution environment with threads like "Daemon Thread [AnyLogic presentation frame manager] (Running)", "Thread [DestroyJavaVM] (Running)", "Thread [AnyLogic simulation performance monitor] (Running)", and "Thread [AWT-EventQueue-0] (Running)".
- Variables/Breakpoints/Expressions:** The Breakpoints tab is active, showing a list of breakpoints. A breakpoint is set at line 10 of Person.java for the method "PerformBirth()".
- Person.java Editor:** The source code for Person.java is visible. A red arrow points to the first line of the "PerformBirth()" method, which is highlighted with a dapped/stippled background. A red text overlay reads: "Double-click in dapped/stippled area on line Where want to stop execution".
- Outline/Projects:** Shows the class hierarchy and a list of methods in Person.java, including "drawModelElements\_PlainVariables\_xjal(Pa)", "drawModelElements\_Statecharts\_xjal(Pan)", "enterState(short, boolean) : void", "EstablishOffspringConnectionsBasedOnMo", "EstablishOffspringLocationBasedOnMother", "evaluateRateOf(TransitionRate) : double", "evaluateTimeoutOf(TransitionTimeout) : d", "executeActionOf(Statechart) : void", "executeActionOf(TransitionMessage, Obj", "executeActionOf(TransitionRate) : void", "executeActionOf(TransitionTimeout) : void", "exitState(short, Transition, boolean, State", "FertilityRateAgeSexEthnicity(double, Sex,", "FinalizeDeath() : void", "get\_Main() : Main", "getNameOf(Statechart) : String", "getNameOf(TransitionMessage) : String", "getNameOf(TransitionRate) : String", "getNameOf(TransitionTimeout) : String", "getNameOfState(short) : String", "getPersistentShape(int) : Object", "getStatechartOf(TransitionMessage) : Sta", "getStatechartOf(TransitionRate) : Statech", "getStatechartOf(TransitionTimeout) : Stat", "IsInfected() : boolean", "IsInReproductiveYears(double) : boolean", "onChange() : void", "onChange\_ethnicity() : void", "onChange\_initialAge() : void", "onChange\_isInitiallyInfected() : void", "onChange\_sex() : void", "onClickModelAt(Panel, double, double, int,", "onClickModelAt\_Parameters\_xjal(Panel, dc", "onClickModelAt\_PlainVariables\_xjal(Panel,", "onDestroy() : void", "onReceive(Object, AgentContinuous2D) : ", "PerformBirth() : void".

The bottom status bar shows "Read-Only", "Smart Insert", "525 : 1", and the system clock "12:52 PM 5/16/2012".

# Return to AnyLogic & Start Simulation via Button Push







# Can Single Step, Explore & Modify Variable Contents, etc.

The screenshot displays the Eclipse IDE interface during a debug session. The main window shows the source code of `Person.java` with the `PerformBirth()` method highlighted. The `Console` view at the bottom shows the following output:

```
void PerformBirth() {  
    Person mother = this;  
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());  
    println("A baby has been born! Baby's id is " + offspring + " while the mother is " + this);  
    // establish connections of infant  
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);  
}
```

The `Variables` view shows the state of the object:

Name	Value
this	Person (id=45)
mother	Person (id=45)
a	Engine (id=47)
a	EnvironmentContinuous2D (id=56)
appearanceTime	0.0
b	Main (id=60)
b	LinkedList<E> (id=61)
c	null
circlesize	10.0
color	Color (id=72)
d	Main\$_Population_Class (id=76)
d	66.6360863667524
Delivery	TransitionTimeout (id=46)
e	null
e	239.18606932080488
ethnicity	Person\$Ethnicity (id=82)

The `Outline` view on the right shows the class structure of `Person`, including methods like `PerformBirth()`, `get_Main()`, `add_Population()`, and `EstablishOffspringConnectionsBasedOnMothersConnections()`.

Warning: Breakpoints are Not  
Shown in Source Window – Just in  
“Breakpoints” area

# Press “Resume” to Continue – Awaiting a Breakpoint

The screenshot displays the Eclipse IDE in a debug state. The top toolbar features a 'Resume (F8)' button, indicating the application is paused. The 'Debug Console' shows the JVM and various threads, including 'Main.TriggerDebugger()' at line 441. The 'Variables' view lists several breakpoints for the 'Main' and 'Person' classes. The 'Outline' view shows the class hierarchy for 'ShapeGroup'. The main editor shows the source code of 'Person.java' with the 'PerformBirth' method highlighted.

```
void PerformBirth (...) {  
    Person mother = this;  
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());  
    println("A baby has been born! Baby's id is " + offspring + " while the mother is " + this);  
    // establish connections of infant  
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);  
    // now position the baby to be close to the mother (otherwise leads to stretching of mother's connections ac  
    EstablishOffspringLocationBasedOnMothersLocation(offspring, mother);  
}
```

# Example Breakpoint in Main

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar shows standard IDE icons, and the menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The main workspace is divided into several panels:

- Debug Console:** Shows the execution stack. The top entry is "Thread [AnyLogic model execution thread] (Suspended (breakpoint at line 345 in Main))". Below it, the stack trace includes:
  - Main.add\_Population(double, Person\$Ethnicity, Person\$Sex, boolean) line: 345
  - Main.executeActionOf(EventRate) line: 289
  - EventRate.execute() line: not available
  - Engine.h() line: not available
  - Engine.a(Engine) line: not available
  - Engine\$.run() line: not available
- Breakpoints:** A list of breakpoints is shown, with "Main [line: 345] - Main" selected and checked.
- Code Editor:** The source code for Main.java is visible. A breakpoint is set on line 345, which corresponds to the line: `Person object = instantiate_Population_xjal(index);`. The code snippet shown is:

```
... * @return newly created embedded object *  
... */  
public Person add_Population( double InitialAge, Person.Ethnicity ethnicity, Person.Sex sex, boolean isInit  
int index = Population.size();  
Person object = instantiate_Population_xjal(index);  
// setup parameters;  
object.InitialAge = InitialAge;  
object.ethnicity = ethnicity;  
object.sex = sex;  
object.isInitiallyInfected = isInitiallyInfected;  
// finish embedded object creation  
create_Population_xjal(object, index);  
object.start();  
return object;  
}
```
- Outline:** A list of methods from the class is shown, including `add_Population(double, Ethnicity, Sex, b...`, `create(): void`, `create_Population_xjal(Person, int): void`, `drawModelElements(Pan`, `evaluateRateOf(EventRate): double`, `evaluateTimeoutOf(EventTimeout): dou`, `executeActionOf(EventRate): void`, `executeActionOf(EventTimeout): void`, `getEmbeddedObjects(): List<Object>`, `getFirstOccurrenceTime(EventTimeout):`, `getModeOf(EventTimeout): int`, `getNameOf(ActiveObject): String`, `getNameOf(ActiveObjectCollection<?>)`, `getNameOf(EventRate): String`, `getNameOf(EventTimeout): String`, `getNameOfShape(int): String`, `getPersistentShape(int): Object`, `getShapeEmbeddedObject(int): Object`, `getShapeReplication(int): int`, `getShapeType(int): int`, `getShapeX(int, int): double`, `getShapeY(int, int): double`, `instantiate_Population_xjal(int): Person`, `onChange(): void`, `onChange_immigrantsPerYear(): void`, `onChange_initialPrevalenceOfInfection()`, `onChange_MeanLifespan(): void`, `onChange_offspringDistanceFromMothe`, `onChange_populationSize(): void`, `onChange_prevalenceOfInfectionAmong`, `onClickModelAt(Pan`, `double, double, i`, `onDestroy(): void`, `onStartup(): void`, `remove_Population(Person): boolean`, `set_immigrantsPerYear(double): void`, and `set_initialPrevalenceOfInfection(double)`.

# Example Breakpoint in Person

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar shows the 'Debug' button is active. The 'Debug Console' on the left shows the execution stack, with the current thread being 'Person.PerformBirth() line: 518'. The 'Variables' view below it lists several variables, including 'Person [line: 518] - Person'. The 'Breakpoints' view shows a breakpoint is set at line 518 in Person.java. The 'Outline' view on the right shows the class structure of 'Person', with 'PerformBirth()' highlighted. The main editor window shows the source code of 'Person.java', with the 'PerformBirth()' method selected. The code includes a call to 'EstablishOffspringConnectionsBasedOnMothersConnections()' and a call to 'EstablishOffspringLocationBasedOnMothersLocation()'.

```
void PerformBirth() {  
    Person mother = this;  
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());  
    println("A baby has been born! Baby's id is " + offspring + " while the mother is " + this);  
    // establish connections of infant  
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);  
    // now position the baby to be close to the mother (otherwise leads to stretching of mother's connections ac  
    EstablishOffspringLocationBasedOnMothersLocation(offspring, mother);  
}
```

# Once at Breakpoint, Can Look at Variables, Single Step, etc.

Debug - C:\Users\Nate\AnyLogicWorkspace\EclipseDebuggingExample\_BUILD\src.generated\abmmmodelwithbirthdeath\Person.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Debug

Thread [AWT-Shutdown] (Running)  
Daemon Thread [AWT-Windows] (Running)  
Daemon Thread [AnyLogic model execution thread] (Suspended (breakpoint at line 518 in Person))  
Person.PerformBirth() line: 518  
Person.executeActionOf(TransitionTimeout) line: 333  
TransitionTimeout.execute() line: not available  
Engine.h() line: not available  
Engine.a(Engine) line: not available  
EngineSa.run() line: not available

Step Over (F6)

Variables Breakpoints Expressions

Main [line: 73] - Main  
Main [line: 76] - Main  
Main [line: 323] - Main  
Main [line: 2421] - Main  
MainClass [line: 12] - main(String[])  
Person [line: 518] - Person  
Person [line: 520] - Person  
PodSchedule [line: 293] - PodSchedule

MainClass.java DefaultTracingFilter MainClass.java DefaultTracingFilter Person.java Person.java

```
void PerformBirth(..) {  
    Person mother = this;  
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());  
    println("A baby has been born! Baby's id is " + offspring + " while the mother is " + this);  
    // establish connections of infant  
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);  
    // now position the baby to be close to the mother (otherwise leads to stretching of mother's connections ac  
    EstablishOffspringLocationBasedOnMothersLocation(offspring, mother);  
}  
void EstablishOffspringConnectionsBasedOnMothersConnections( Person offspring, Person mother ) {
```

Outline

new ShapeGroup() { ... }  
CurrentAge() : double  
drawModelElements(Panel, Graphics2D, ...)  
enterState(short, boolean) : void  
EstablishOffspringConnectionsBasedOnMothersConnections( ... )  
EstablishOffspringLocationBasedOnMothersLocation( ... )  
evaluateRateOf(TransitionRate) : double  
evaluateTimeoutOf(TransitionTimeout) : double  
executeActionOf(Statechart) : void  
executeActionOf(TransitionMessage, Object) : void  
executeActionOf(TransitionRate) : void  
executeActionOf(TransitionTimeout) : void  
exitState(short, Transition, boolean, Statechart) : void  
FertilityRateAgeSexEthnicity(double, Sex, ...)  
FinalizeDeath() : void  
get\_Main() : Main  
getNameOf(Statechart) : String  
getNameOf(TransitionMessage) : String  
getNameOf(TransitionRate) : String  
getNameOf(TransitionTimeout) : String  
getNameOfState(short) : String  
getPersistentShape(int) : Object  
getStatechartOf(TransitionMessage) : Statechart  
getStatechartOf(TransitionRate) : Statechart  
getStatechartOf(TransitionTimeout) : Statechart  
IsInfected() : boolean  
isInReproductiveYears(double) : boolean  
onChange() : void  
onChange\_ethnicity() : void  
onChange\_initialAge() : void  
onChange\_isInitiallyInfected() : void  
onChange\_sex() : void  
onClickModelAt(Panel, double, double, double) : void  
onDestroy() : void  
onReceive(Object, Agent) : void  
PerformBirth() : void

Read-Only Smart Insert 518 : 22 Build Project

# Variables Displayed

The screenshot shows the Eclipse IDE in a debug state. The top toolbar includes icons for file operations, debugging, and navigation. The 'Debug' console on the left shows the execution stack, with the current frame being `Person.PerformBirth()` at line 520. The 'Variables' view in the center displays the following data:

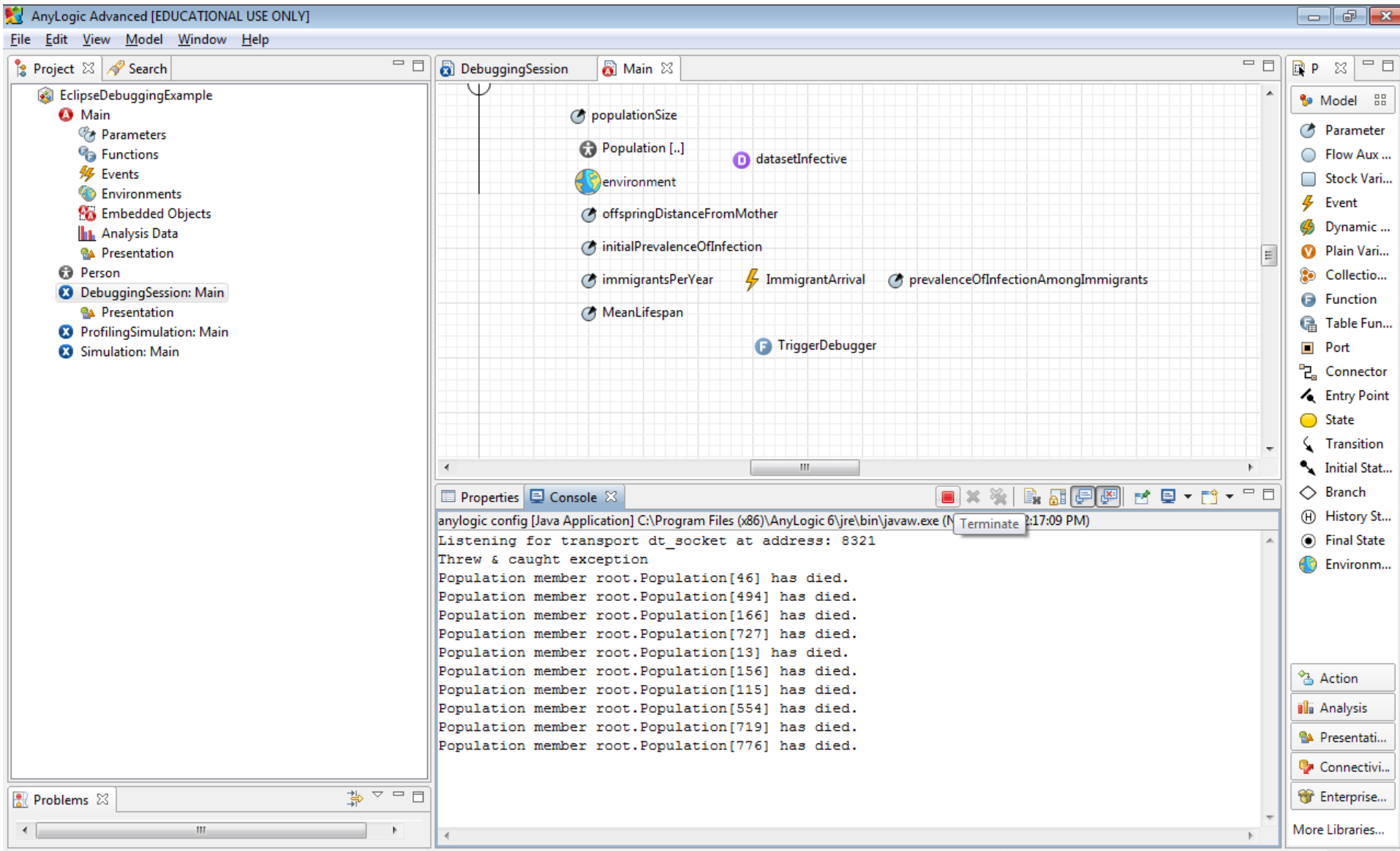
Name	Value
this	Person (id=61)
mother	Person (id=61)
offspring	Person (id=64)

The bottom editor shows the source code of `Person.java`, with the `PerformBirth()` method selected. The code includes a `println` statement that has been executed, displaying the message: "A baby has been born! Baby's id is 64 while the mother is 61". The right-hand side of the IDE shows the 'Outline' view, listing the methods of the `new ShapeGroup()` class, including `PerformBirth()`.

1:1 Build Project



# Terminating Execution from AnyLogic Console



The screenshot displays the AnyLogic Advanced [EDUCATIONAL USE ONLY] interface. The main workspace shows a debugging session for a model named 'EclipseDebuggingExample'. The 'Main' component is selected, and its internal elements are visible, including 'populationSize', 'Population [..]', 'environment', 'datasetInfective', 'offspringDistanceFromMother', 'initialPrevalenceOfInfection', 'immigrantsPerYear', 'ImmigrantArrival', 'prevalenceOfInfectionAmongImmigrants', 'MeanLifespan', and 'TriggerDebugger'.

The console window at the bottom shows the following output:

```
anylogic config [Java Application] C:\Program Files (x86)\AnyLogic 6\jre\bin\javaw.exe (N Terminate 2:17:09 PM)
Listening for transport dt_socket at address: 8321
Threw & caught exception
Population member root.Population[46] has died.
Population member root.Population[494] has died.
Population member root.Population[166] has died.
Population member root.Population[727] has died.
Population member root.Population[13] has died.
Population member root.Population[156] has died.
Population member root.Population[115] has died.
Population member root.Population[554] has died.
Population member root.Population[719] has died.
Population member root.Population[776] has died.
```

The 'Terminate' button is highlighted over the console output, indicating the process of ending the simulation.

# Eclipse is Now Detached

The screenshot displays the Eclipse IDE interface during a debug session. The title bar indicates the current file is `Person.java` in the `src.generated\abmmmodelwithbirthdeath` package. The menu bar includes `File`, `Edit`, `Source`, `Refactor`, `Navigate`, `Search`, `Project`, `Run`, `Window`, and `Help`. The toolbar contains various icons for file operations and debugging.

The **Debug** console shows the following output:

```
<terminated> Anylogic Application [Remote Java Application]
<disconnected> Java HotSpot(TM) Client VM[localhost:8321]
```

The **Outline** view on the right lists the methods of the `new ShapeGroup() { ...}` class:

- `CurrentAge() : double`
- `drawModelElements(Panel, Graphics2D)`
- `enterState(short, boolean) : void`
- `EstablishOffspringConnectionsBasedOnM...`
- `EstablishOffspringLocationBasedOnMotl...`
- `evaluateRateOf(TransitionRate) : double`
- `evaluateTimeoutOf(TransitionTimeout) :`
- `executeActionOf(Statechart) : void`
- `executeActionOf(TransitionMessage, Ob...`
- `executeActionOf(TransitionRate) : void`
- `executeActionOf(TransitionTimeout) : vc...`
- `exitState(short, Transition, boolean, Sta...`
- `FertilityRateAgeSexEthnicity(double, Sex,`
- `FinalizeDeath() : void`
- `get_Main() : Main`
- `getNameOf(Statechart) : String`
- `getNameOf(TransitionMessage) : String`
- `getNameOf(TransitionRate) : String`
- `getNameOf(TransitionTimeout) : String`
- `getNameOfState(short) : String`
- `getPersistentShape(int) : Object`
- `getStatechartOf(TransitionMessage) : Sta...`
- `getStatechartOf(TransitionRate) : Statech...`
- `getStatechartOf(TransitionTimeout) : Sta...`
- `IsInfected() : boolean`
- `isInReproductiveYears(double) : boolean`
- `onChange() : void`
- `onChange_ethnicity() : void`
- `onChange_InitialAge() : void`
- `onChange_isInitiallyInfected() : void`
- `onChange_sex() : void`
- `onClickModelAt(Panel, double, double, i...`
- `onDestroy() : void`
- `onReceive(Object, Agent) : void`
- `PerformBirth() : void`

The **Variables** view is currently empty. The **Breakpoints** and **Expressions** views are also empty.

The **Source** editor shows the following code in `Person.java`:

```
void PerformBirth( ) {
    Person mother = this;
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());
    println("A baby has been born! - Baby's id is " + offspring + " while the mother is " + this);
    // establish connections of infant
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);
    // now position the baby to be close to the mother (otherwise leads to stretching of mother's connections ac
    EstablishOffspringLocationBasedOnMothersLocation(offspring, mother);
}

void EstablishOffspringConnectionsBasedOnMothersConnections( Person offspring, Person mother ) {
```

The bottom status bar shows the zoom level as `1:1` and the current project as `Build Project`.

# Remembering Breakpoints

- Note Eclipse *does* remember breakpoints from session to session
- So breakpoints that set earlier in an anylogic session will work again even after close eclipse and restart it again
- Suggestions
  - Consider creating a common breakpoints (e.g. at Main.start)
  - Disable and enable breakpoints rather than deleting them

# Example of Debugging Session

Debug - U:\Classes\371\_Spring2009\Project\Deliverable 4\Milestone4\newDev\newDev\src\oneoak\digdog\gui\MainWindow.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Debug

Anylogic Application [Remote Java Application]

- Java HotSpot(TM) Client VM[localhost:8321] (Suspended)
  - Daemon Thread [AnyLogic model execution thread] (Suspended)
    - FileOutputStream.writeBytes(byte[], int, int) line: not available [native method]
    - FileOutputStream.write(byte[], int, int) line: not available
    - BufferedOutputStream.flushBuffer() line: not available
    - BufferedOutputStream.flush() line: not available
    - PrintStream.write(byte[], int, int) line: not available
    - StreamEncoder.writeBytes() line: not available
    - StreamEncoder.implFlushBuffer() line: not available
    - StreamEncoder.flushBuffer() line: not available
    - OutputStreamWriter.flushBuffer() line: not available
    - PrintStream.write(String) line: not available
    - PrintStream.print(String) line: not available
    - PrintStream.println(Object) line: not available
    - Utilities.traceln(Object) line: not available
    - Person.PerformBirth() line: 520
    - Person.executeActionOf(TransitionTimeout) line: 333
    - TransitionTimeout.execute() line: not available
    - Engine.h() line: not available
    - Engine.a(Engine) line: not available
    - Engine\$.run() line: not available
  - Daemon System Thread [TimerQueue] (Suspended)
  - Thread [DestroyJavaVM] (Suspended)
  - Daemon Thread [AnyLogic presentation frame manager] (Suspended)
  - Thread [AnyLogic simulation performance monitor] (Suspended)
  - Thread [AWT-EventQueue-0] (Suspended)
  - Thread [AWT-Shutdown] (Suspended)
  - Daemon Thread [AWT-Windows] (Suspended)
  - Daemon System Thread [Java2D Disposer] (Suspended)
  - Daemon System Thread [Attach Listener] (Suspended)
  - Daemon System Thread [Signal Dispatcher] (Suspended)
  - Daemon System Thread [Finalizer] (Suspended)
  - Daemon System Thread [Reference Handler] (Suspended)
  - Daemon Thread [Image Fetcher 0] (Suspended)

# Another Route: Catching Exceptions at Defined Places of Interest

# Example Setup: Set up Function to Trigger the Debugger

The screenshot displays the AnyLogic Advanced software interface, specifically the 'DebuggingSession' window. The main workspace shows a grid of model components, including 'populationSize', 'Population [..]', 'datasetInfective', 'environment', 'offspringDistanceFromMother', 'initialPrevalenceOfInfection', 'immigrantsPerYear', 'ImmigrantArrival', 'prevalenceOfInfectionAmongImmigrants', and 'MeanLifespan'. A function named 'TriggerDebugger' is highlighted in the workspace.

The 'TriggerDebugger' function is defined in the 'Code' tab of the Properties window. The function body is as follows:

```
try
{
    throw new RuntimeException("arbitrary");
}
catch (RuntimeException e)
{
    traceIn("Threw & caught exception");
}
```

The interface also shows a 'Project' view on the left with a tree structure for 'EclipseDebuggingExample\*' containing 'Main', 'Parameters', 'Functions', 'Events', 'Environments', 'Embedded Objects', 'Analysis Data', 'Presentation', 'Person', 'DebuggingSession: Main', 'ProfilingSimulation: Main', and 'Simulation: Main'. The 'Console' window is currently empty, and the 'Problems' window is also empty.

# In Startup Code for Model, Call Function

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a model diagram with various components like 'populationSize', 'Population [...]', 'environment', 'datasetInfective', 'offspringDistanceFromMother', 'initialPrevalenceOfInfection', 'immigrantsPerYear', 'ImmigrantArrival', 'prevalenceOfInfectionAmongImmigrants', and 'MeanLifespan'. A 'TriggerDebugger' component is also visible.

The Properties window is open for the 'Main - Active Object Class'. The 'Startup Code' field contains the following code:

```
environment.deliverToRandom("Infect!");  
TriggerDebugger();
```

The 'Destroy Code' field is currently empty.

The interface also shows a Project Explorer on the left with a tree view of the model structure, including 'Main', 'Parameters', 'Functions', 'Events', 'Environments', 'Embedded Objects', 'Analysis Data', 'Presentation', 'Person', 'DebuggingSession: Main', 'ProfilingSimulation: Main', and 'Simulation: Main'. The right sidebar contains a palette of model components such as 'Model', 'Parameter', 'Flow Aux...', 'Stock Vari...', 'Event', 'Dynamic...', 'Plain Vari...', 'Collectio...', 'Function', 'Table Fun...', 'Port', 'Connector', 'Entry Point', 'State', 'Transition', 'Initial Stat...', 'Branch', 'History St...', 'Final State', and 'Environ...'. At the bottom right, there are buttons for 'Action', 'Analysis', 'Presentati...', 'Connectivi...', and 'Enterprise...', along with a 'More Libraries...' link.

# Request Creation of Exception Breakpoint

The screenshot displays the Eclipse IDE interface with the 'Run' menu open. The 'Add Java Exception Breakpoint...' option is highlighted. The background shows the 'Person.java' file in the editor, with a line of code selected: `is " + offspring + " while the mother is " + this);`. The 'Outline' view on the right lists the methods of the `new ShapeGroup()` class, including `CurrentAge() : double`, `drawModelElements(Panels, Graphics2D)`, `enterState(short, boolean) : void`, `establishOffspringConnectionsBasedOnMot`, `establishOffspringLocationBasedOnMot`, `evaluateRateOf(TransitionRate) : double`, `evaluateTimeoutOf(TransitionTimeout) :`, `executeActionOf(Statechart) : void`, `executeActionOf(TransitionMessage, Ob`, `executeActionOf(TransitionRate) : void`, `executeActionOf(TransitionTimeout) : vc`, `exitState(short, Transition, boolean, State`, `FertilityRateAgeSexEthnicity(double, Sex,`, `FinalizeDeath() : void`, `get_Main() : Main`, `getNameOf(Statechart) : String`, `getNameOf(TransitionMessage) : String`, `getNameOf(TransitionRate) : String`, `getNameOf(TransitionTimeout) : String`, `getNameOfState(short) : String`, `getPersistentShape(int) : Object`, `getStatechartOf(TransitionMessage) : Sta`, `getStatechartOf(TransitionRate) : Statech`, `getStatechartOf(TransitionTimeout) : Sta`, `isInfected() : boolean`, `isInReproductiveYears(double) : boolean`, `onChange() : void`, `onChange_ethnicity() : void`, `onChange_initialAge() : void`, `onChange_sex() : void`, `onClickModelAt(Panels, double, double, i`, `onDestroy() : void`, `onReceive(Object, Agent) : void`, and `PerformBirth() : void`.

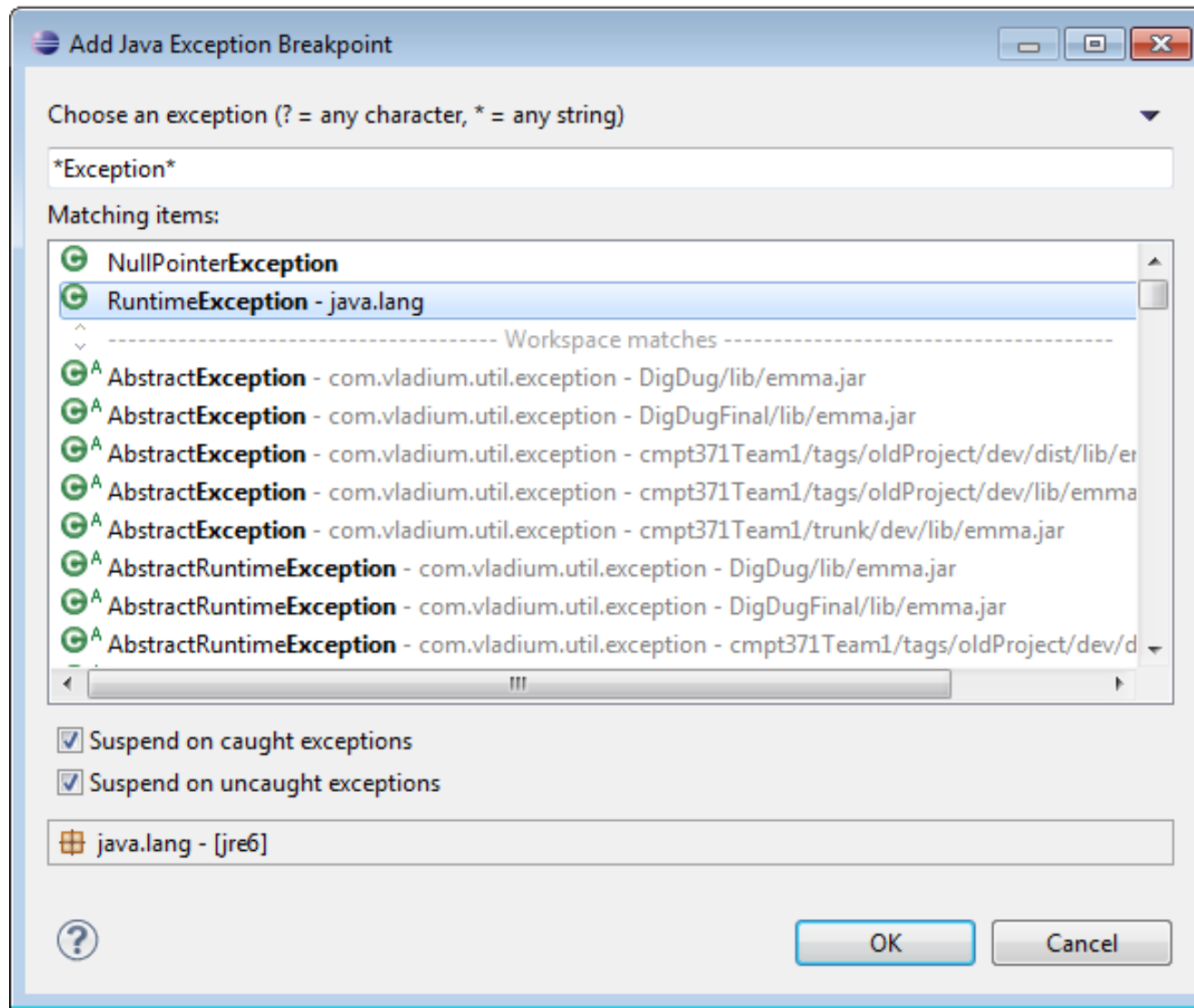
The 'Run' menu options include:

- Resume
- Suspend
- Terminate
- Step Into
- Step Over
- Step Return
- Run to Line (Shift+F5)
- Use Step Filters
- Run (Ctrl+F11)
- Debug (F11)
- Profile
- Profile History
- Profile As
- Profile Configurations...
- Run History
- Run As
- Run Configurations...
- Debug History
- Debug As
- Debug Configurations...
- Toggle Breakpoint (Ctrl+Shift+B)
- Toggle Line Breakpoint
- Toggle Method Breakpoint
- Toggle Watchpoint
- Skip All Breakpoints
- Remove All Breakpoints
- Add Java Exception Breakpoint... (highlighted)
- Add Class Load Breakpoint...
- All References...
- All Instances... (Ctrl+Shift+N)
- Watch
- Inspect (Ctrl+Shift+I)

The status bar at the bottom shows 'Writable', 'Smart Insert', '520 : 1', and 'Build Project'.



# Request as Breakpoint Regardless of Handling



# Should Now be in List of Enabled Breakpoints

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes standard IDE actions like Run, Stop, and Step Over. The main editor displays the source code of `Person.java`, with a breakpoint set on the line `EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);`. The Breakpoints view on the left shows this breakpoint is active. The Outline view on the right lists the methods of the `new ShapeGroup()` class, including `PerformBirth()`. The status bar at the bottom indicates the project is being built.

Debug - C:\Users\Nate\AnyLogicWorkspace\EclipseDebuggingExample\_BUILD\src.generated\abmmmodelwithbirthdeath\Person.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Debug <terminated> Anylogic Application [Remote Java Application]  
<disconnected> Java HotSpot(TM) Client VM[localhost:8321]

Breakpoints

- RuntimeException
- NullPointerException: caught and uncaught
- RuntimeException: caught and uncaught
- CreateScenarioDialog [line: 436] - new Anonymous
- HTMLLinksToFiles [line: 27] - main(String[])
- HTMLLinksToFiles [line: 29] - main(String[])

MainClass.java DefaultTracingFilter MainClass.java DefaultTracingFilter Main.java Person.java

```
void PerformBirth(..) {  
    Person mother = this;  
    Person offspring = get_Main().add_Population((double) 0, ethnicity, RandomSex(), this.IsInfected());  
    println("A baby has been born! Baby's id is " + offspring + " while the mother is " + this);  
    // establish connections of infant  
    EstablishOffspringConnectionsBasedOnMothersConnections(offspring, mother);  
    // now position the baby to be close to the mother (otherwise leads to stretching of mother's connections across the map)  
    EstablishOffspringLocationBasedOnMothersLocation(offspring, mother);  
}  
  
void EstablishOffspringConnectionsBasedOnMothersConnections(Person offspring, Person mother) {  
    // now establish links between the baby and all of the mother's connections  
}
```

new ShapeGroup() { ... }

- CurrentAge(): double
- drawModelElements(Panel, Graphics2D, boolean): void
- enterState(short, boolean): void
- EstablishOffspringConnectionsBasedOnMothersConnections(Person, Person): void
- EstablishOffspringLocationBasedOnMothersLocation(Person, Person): void
- evaluateRateOf(TransitionRate): double
- evaluateTimeoutOf(TransitionTimeout): double
- executeActionOf(Statechart): void
- executeActionOf(TransitionMessage, Object): void
- executeActionOf(TransitionRate): void
- executeActionOf(TransitionTimeout): void
- exitState(short, Transition, boolean, Statechart): void
- FertilityRateAgeSexEthnicity(double, Sex, boolean): void
- FinalizeDeath(): void
- get\_Main(): Main
- getNameOf(Statechart): String
- getNameOf(TransitionMessage): String
- getNameOf(TransitionRate): String
- getNameOf(TransitionTimeout): String
- getNameOfState(short): String
- getPersistentShape(int): Object
- getStatechartOf(TransitionMessage): Statechart
- getStatechartOf(TransitionRate): Statechart
- getStatechartOf(TransitionTimeout): Statechart
- IsInfected(): boolean
- isInReproductiveYears(double): boolean
- onChange(): void
- onChange\_ethnicity(): void
- onChange\_initialAge(): void
- onChange\_isInitiallyInfected(): void
- onChange\_sex(): void
- onClickModelAt(Panel, double, double, double): void
- onDestroy(): void
- onReceive(Object, Agent): void
- PerformBirth(): void

Build Project

# Back in Eclipse, the Debugger Should have been Triggered & at Exception Handler

(If not, close "Main.java" and double-click on topmost "stack frame" (Where Exception is triggered

The screenshot displays the Eclipse IDE interface during a debugging session. The top toolbar includes the 'Debug' button. The 'Debug' console shows a list of threads, with 'Main.TriggerDebugger()' at line 441 selected. The 'Variables' view shows a 'RuntimeException' object. The 'Main.java' editor shows the source code with a 'throw new RuntimeException(\"arbitrary\");' statement highlighted. The 'Outline' view shows the class structure with 'TriggerDebugger()' at the bottom.

```
Debug - C:\Users\Nate\AnyLogicWorkspace\EclipseDebuggingExample_BUILD\src.generated\abmmmodelwithbirthdeath\Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Debug
Java HotSpot(TM) Client VM[localhost:8321]
  Thread [DestroyJavaVM] (Running)
  Daemon Thread [AnyLogic presentation frame manager] (Running)
  Thread [AnyLogic simulation performance monitor] (Running)
  Thread [AWT-EventQueue-0] (Running)
  Thread [AWT-Shutdown] (Running)
  Daemon Thread [AWT-Windows] (Running)
  Daemon Thread [AnyLogic ShapeControl action handler] (Suspended (exception RuntimeException))
  Main.TriggerDebugger() line: 441

Variables
[ ] RuntimeException
[ ] NullPointerException: caught and uncaught
[ ] RuntimeException: caught and uncaught
[ ] CreateScenarioDialog [line: 436] - new Anonymous
[ ] HTMLLinksToFiles [line: 27] - main(String[])
[ ] HTMLLinksToFiles [line: 29] - main(String[])
[ ] HTMLLinksToFiles [line: 36] - main(String[])
[ ] HTMLLinksToFiles [line: 38] - main(String[])

MainClass.java DefaultTracingFilter MainClass.java DefaultTracingFilter Person.java Main.java
--// User functions -----
void TriggerDebugger (...) {
    try
    {
        throw new RuntimeException("arbitrary");
    }
    catch (RuntimeException e)
    {
        traceIn("Threw & caught exception");
    }
}

--// Analysis Data Elements
```

# Using the AnyLogic Built-in Debugger

# Running the Debugger

The screenshot displays the AnyLogic Professional software interface. The main window shows a Java code editor with the following code:

```
package sir_agent_based_networks;  
import java.sql.Connection;  
import java.sql.SQLException;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Calendar;  
import java.util.Collection;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.Currency;  
import java.util.Date;  
import java.util.Enumeration;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.Hashtable;  
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.ListIterator;  
import java.util.Locale;  
import java.util.Map;  
import java.util.Random;  
import java.util.Set;
```

A context menu is open over the line `import java.util.HashSet;`, showing two options:

- SIR Agent Based Networks / Simulation
- SIR Agent Based Networks / MonteCarlo

The interface includes several panels:

- Projects:** Shows a tree view of the project structure, including 'SIR Agent Based Networks', 'Main', 'Parameters', 'Functions', 'networkTypeToString', 'Environments', 'Embedded Objects', 'Analysis Data', 'Presentation', 'Person', 'Simulation: Main', and 'MonteCarlo: Main'.
- Palette:** Contains various modeling elements such as 'General', 'Parameter', 'Event', 'Dynamic Event', 'Plain Variable', 'Collection', 'Function', 'Table Function', 'Port', 'Connector', 'Environment', 'System Dynamics', 'Statechart', 'Actionchart', 'Analysis', 'Presentation', '3D', 'Controls', 'Connectivity', 'Pictures', '3D Objects', 'Enterprise Library', 'Pedestrian Library', 'Rail Yard Library', and 'Palettes...'. The 'Environment' element is highlighted.
- Problems:** A table with columns 'Description' and 'Locat..'. It is currently empty.
- Properties and Console:** The console shows the message: `<terminated> anylogic config [Java Application] C:\Program Files (x86)\AnyLogic 6 Professional\jre\bin\javaw.exe (Nov 1, 2010 1:19:13 AM)`

The status bar at the bottom indicates 'Read-Only', 'Smart Insert', and '16 : 26'.

# Running the Models

The screenshot displays the AnyLogic Professional software interface. The main window is titled "SIR Agent Based Simulation - AnyLogic Professional [EVALUATION USE ONLY]". The central area shows the "SIR Agent Based Model of Disease Diffusion" configuration screen. On the left, there is a "Model parameters" section with various input fields and dropdown menus. On the right, there is a "Description" section with text explaining the model's states and transitions. At the bottom, there is a "Run the model" button and a status bar showing simulation progress and resource usage.

**Model parameters**

Total population:	<input type="text" value="200"/>
Fraction initially infected:	<input type="text" value="0.05"/>
Contact rate:	<input type="text" value="5.0"/> contacts per day
Infectivity:	<input type="text" value="0.05"/>
Average illness duration:	<input type="text" value="15.0"/> days
Layout type:	<input type="text" value="Ring"/>
Network type:	<input type="text" value="Random"/>
Links per agent:	<input type="text" value="10"/>
Maximum link distance:	<input type="text" value="50.0"/>
Percent of long distance links:	<input type="text" value="0.05"/>
M:	<input type="text" value="2"/>

**Description**

We distinguish between three different states of a person: Susceptible, Infectious and Recovered. To reflect the fact that some people are already infected at the beginning of the simulation, the agent statechart entry point has a branch InitiallyInfected. The choice of the initial state is probabilistic and is based on a global model parameter PercentInitiallyInfected.

The transition to the state Infectious models the event of disease being passed to the agent from a sick person. In the model terms the trigger of that transition is a message "infection". Once in the Infectious state, the agent is able to pass the disease to others, therefore we are interested in his contacts. We assume that ContactRate is constant while in the Infectious state, and we can use internal transition Contact that will be repeatedly taken until the agent recovers. In the action of that transition the agent chooses another agent from the people that he knows (this is defined by the social network), and sends him the message "infection". However, as not every contact results in infection being passed, message is sent with the probability InfectionProbability, which is yet another global parameter. If the message reaches an agent who is already infectious or recovered, it is ignored. Finally, the transition Recovery is a timeout that should model the illness duration.

This model allows you to explore the dynamics of disease diffusion in these types of network:

- Based on distance – people are linked if the (geographical) distance between them is not more than a given value.
- Random – a person is linked to a random subset of the population.
- Small World (Watts 1999) – most links are neighbors, but there is a certain percent of long-range links. To establish neighborhood people can be placed e.g. on a virtual ring
- Scale Free (Barabasi, Albert 1999) – some people are "hubs" with lots of connections and some are "hermits". This type of network is build using a preferential attachment algorithm where a probability of a new person links to the existing people is proportional to the number of links those people already have.
- Ring Lattice – each person is linked to a fixed number of his closest neighbors on a ring. This type of network is most distant to the fully connected network.

**anylogic config [Java]**

```
package sir_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
import java_...
```

**Projects**

- SIR Agent Based Networks
  - Main
  - Parameters
  - Functions
  - networkTypeTo
  - Environments
  - Embedded Objects
  - Analysis Data
  - Presentation
  - Person
  - Simulation: Main
  - MonteCarlo: Main

**Properties**

anylogic config [Java]

**Run:** 0  Idle  Step: -

**EPS:** 0 **FPS:** 0.0 **Memory:** 10M of 297M

0.0 sec

# Setting a Breakpoint

The screenshot displays the AnyLogic Professional software interface. The main window shows a grid-based workspace with the text "SIR Agent Ba" visible. The interface includes several panels:

- Debug**: A tree view on the left showing various threads like "Thread [AWT-Windows] (Ru)".
- Breakpoints**: A panel at the top showing a breakpoint set on "networkTypeToString : Function - Body [relative line: 1]".
- Variables** and **Expressions**: Panels for monitoring and editing code elements.
- Palette**: A panel on the right containing various components like "Parameter", "Event", "Dynamic Event", etc.
- Properties** and **Console**: Panels at the bottom for viewing object properties and running output.
- Projects**: A panel on the bottom left showing a project tree with "SIR Agent Based Networks" and "Main".

The **Code** panel at the bottom center shows the following function body:

```
Function body:  
switch( type ) {  
  case Environment.NETWORK_USER_DEFINED:  
    return "Custom";  
  case Environment.NETWORK_RANDOM:  
    return "Random";  
  case Environment.NETWORK_ALL_IN_RANGE:  
    return "Based on distance";  
  case Environment.NETWORK_RING_LATTICE:  
    return "Ring lattice";  
  case Environment.NETWORK_SMALL_WORLD:  
    return "Small world";  
  case Environment.NETWORK_SCALE_FREE:  
    return "Scale free";  
  default: return "Unknown";  
}
```

# When we Hit the Breakpoint...

The screenshot displays the AnyLogic Professional software interface during a debugging session. The main window shows the source code of the `networkTypeToString` function in `Main.java`. A breakpoint is set at the beginning of the `switch` statement, and the execution has stopped at this point. The `Variables` window shows the current state of the function, with `this` pointing to the `Main` object and `type` set to `1`. The `Properties` window at the bottom shows the details of the `networkTypeToString` function, including its name, access level, and return type (`String`).

**Breakpoints**

- networkTypeToString : Function - Body [relative line: 1]

**Variables**

- this (sir\_agent\_based\_networks.Main) : (id=46)
- type (int) : 1

```
String
networkTypeToString( int type ) {
switch( type ) {
case Environment.NETWORK_USER_DEFINED:
return "Custom";
case Environment.NETWORK_RANDOM:
return "Random";
case Environment.NETWORK_ALL_IN_RANGE:
return "Based on distance";
case Environment.NETWORK_RING_LATTICE:
return "Ring lattice";
case Environment.NETWORK_SMALL_WORLD:
return "Small world";
case Environment.NETWORK_SCALE_FREE:
return "Scale free";
}
```

**Properties - networkTypeToString - Function**

**General**

Name: networkTypeToString  Show name  Ignore  Public  Show at runtime

**Code**

Access: default  Static

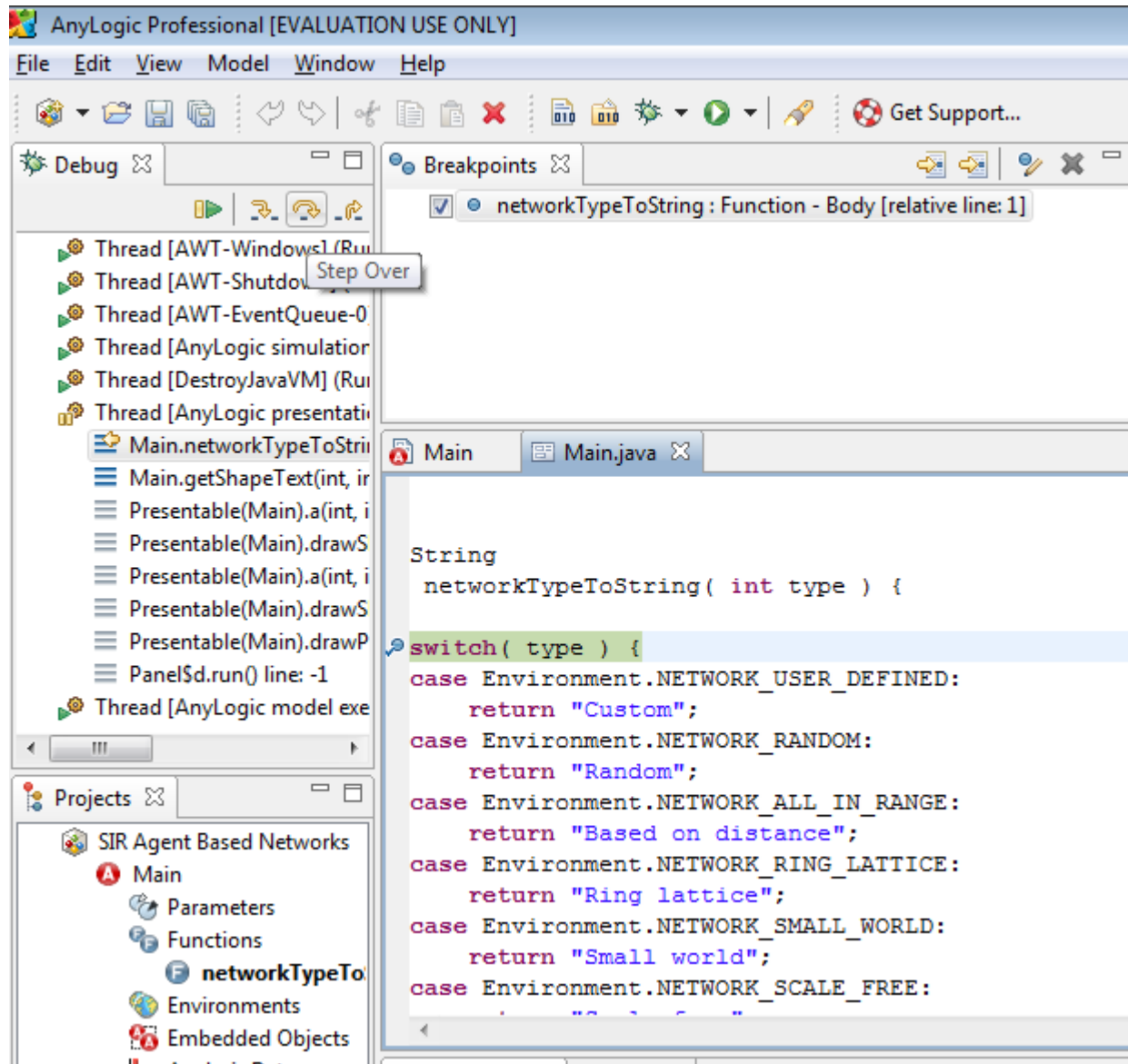
Return type:  void  boolean  int  double  String  Other: String

Function arguments:

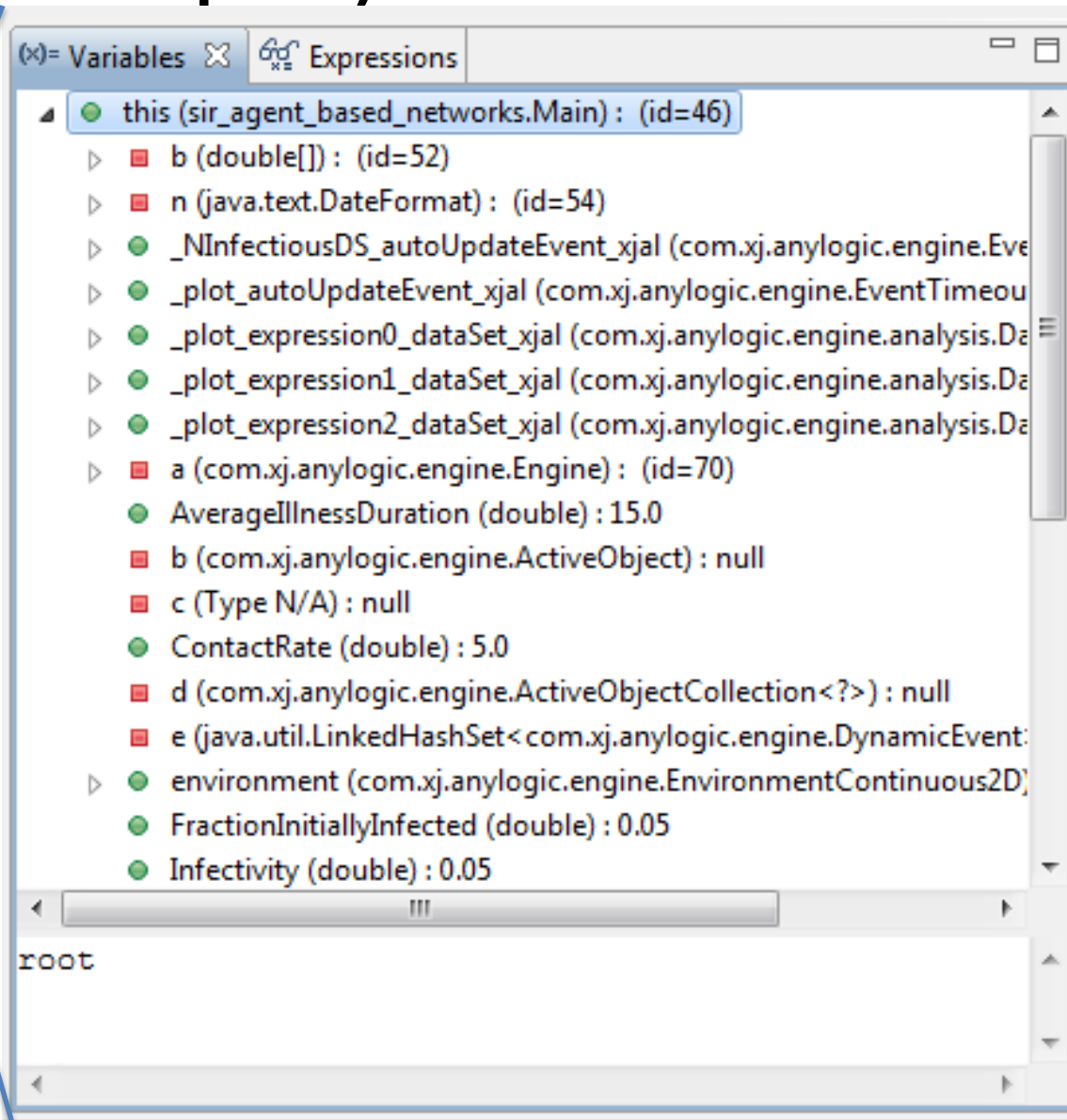
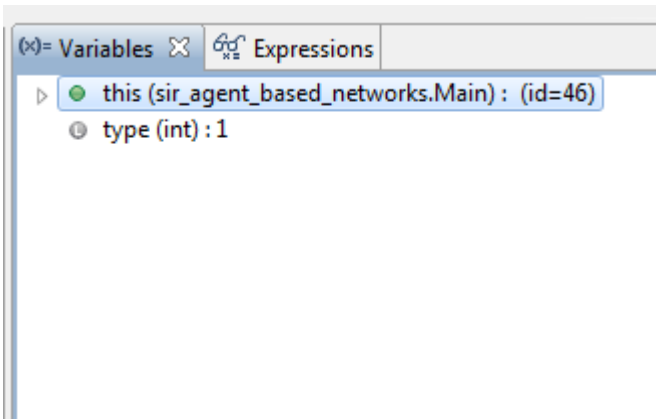
Read-Only Smart Insert 559 : 17



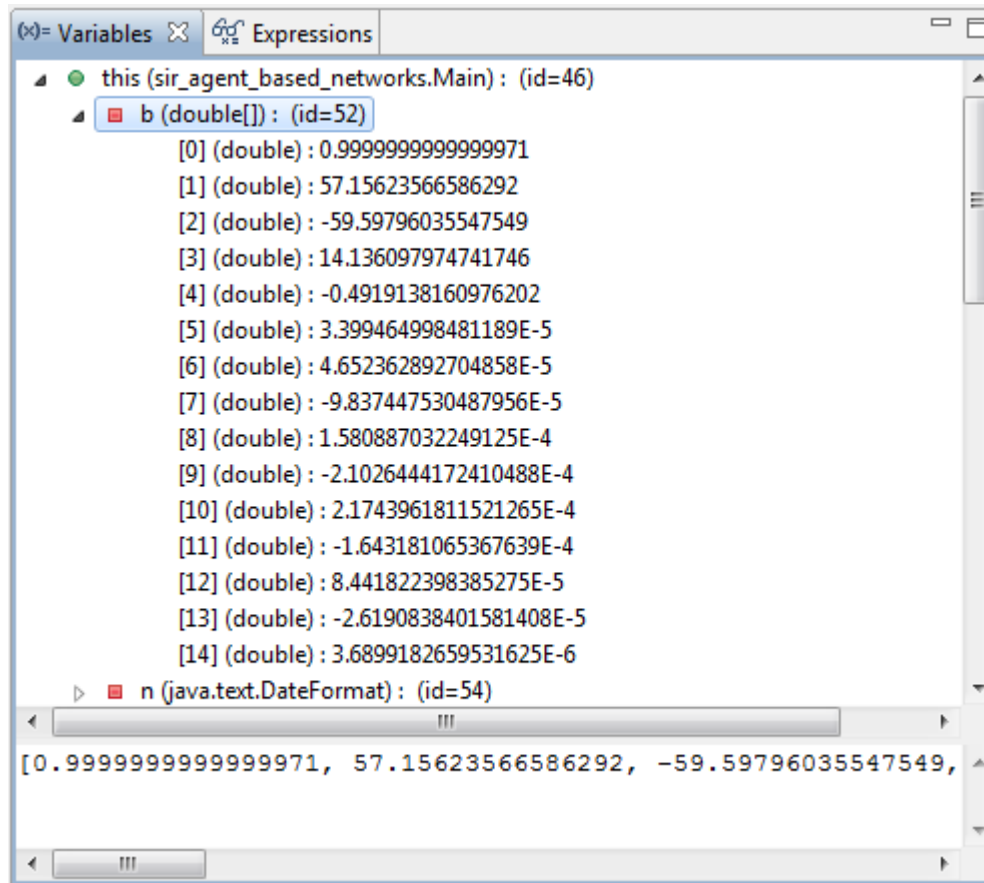
# Components to Direct Execution



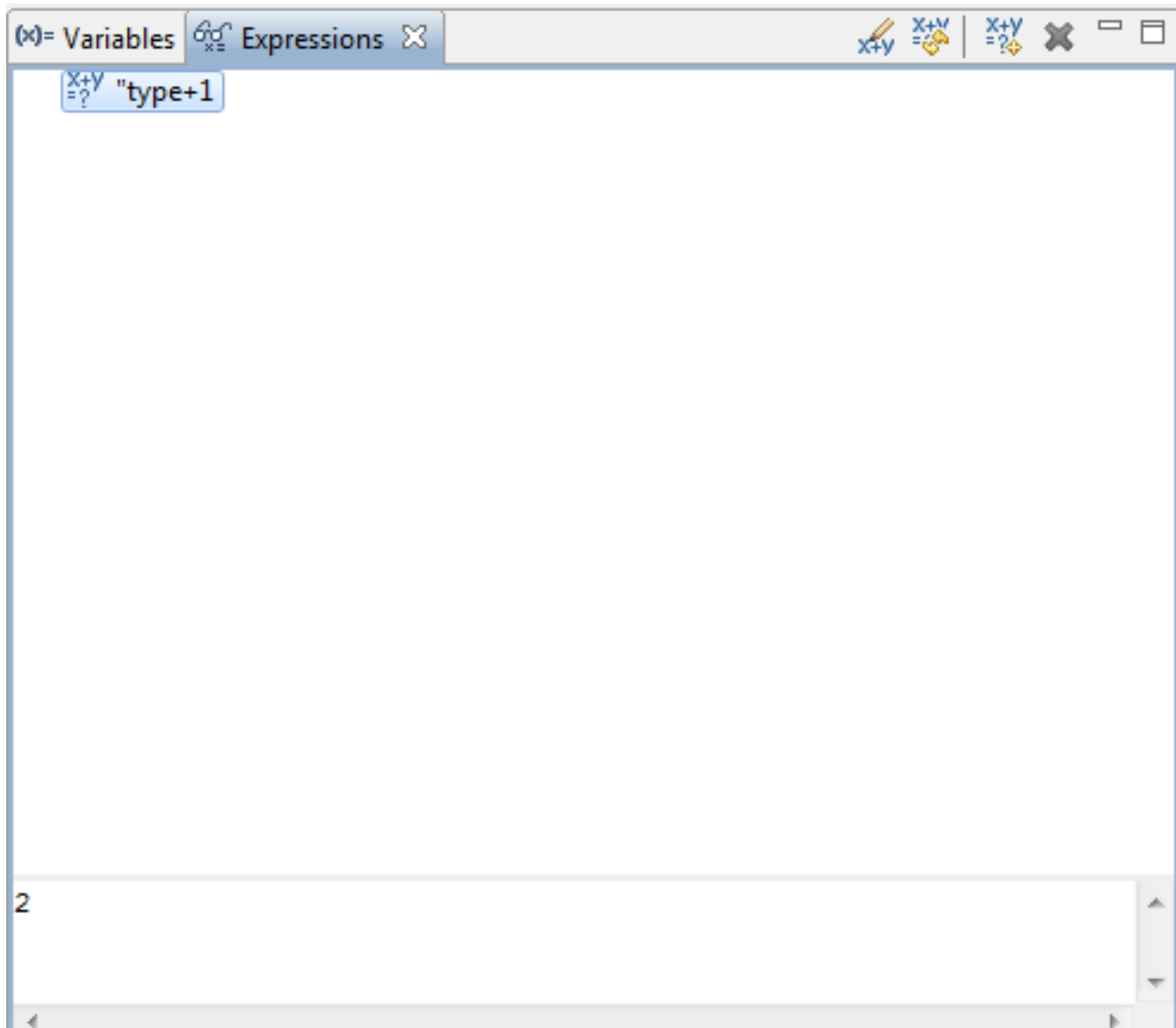
# Visible (“In-Scope”) Variables



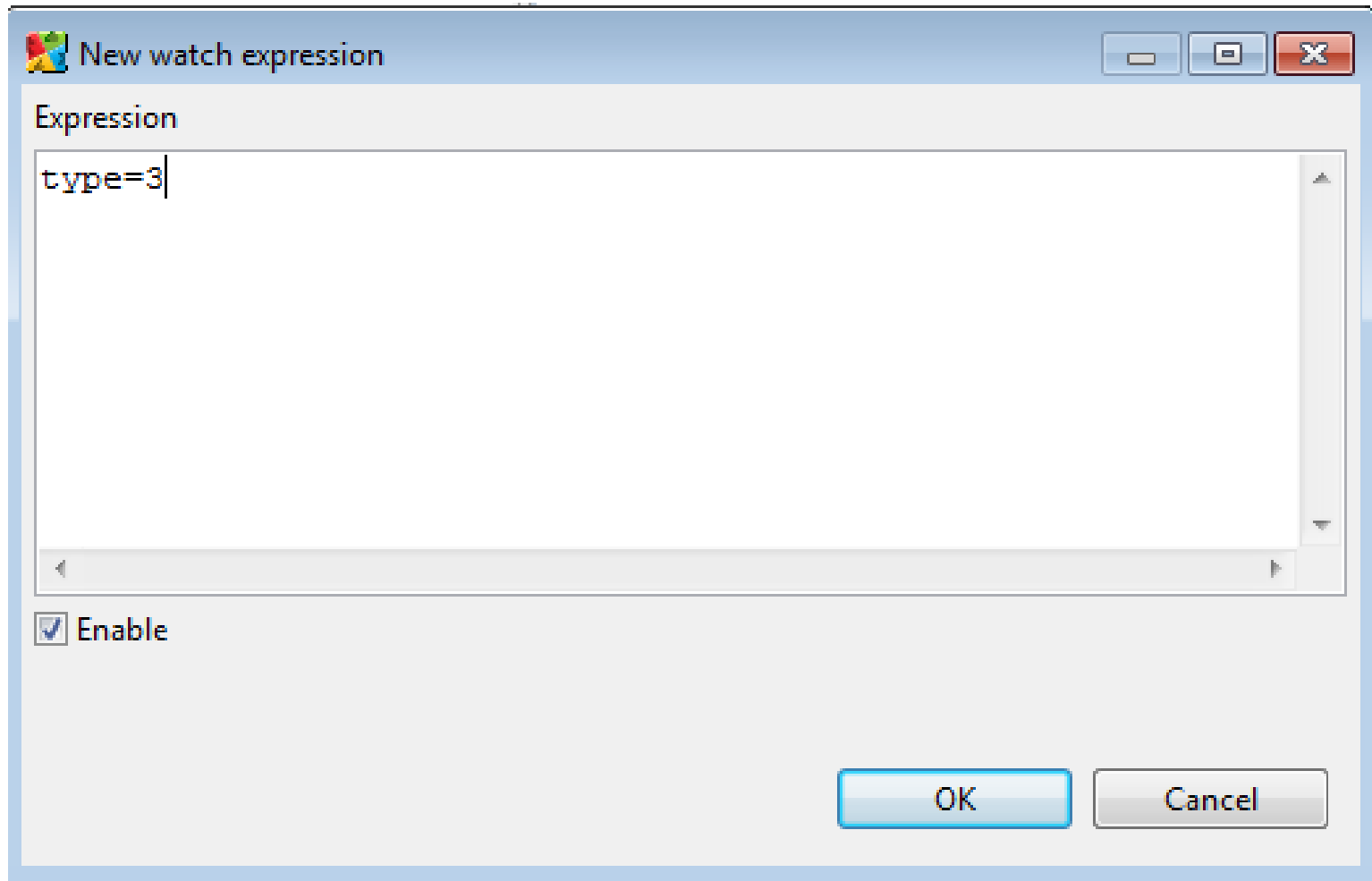
# Exploring Composite Variable Values in the Debugger



# Inspecting Composite Variables



# Changing Variable Values During Debugging



# Stepping into Auto-Generated Code

The screenshot displays the AnyLogic Professional interface during a debugging session. The main window shows the source code of `Main.java` with a breakpoint set on the `networkTypeToString` function call. The `Variables` window shows the current state of the `this` object and its properties. The `Properties` window for the `networkTypeToString` function is also visible.

**Breakpoints:** `networkTypeToString : Function - Body [relative line: 1]`

**Variables:**

- `this (sir_agent_based_networks.Main) : (id=46)`
- `_shape (int) : 15`
- `index (int) : 0`

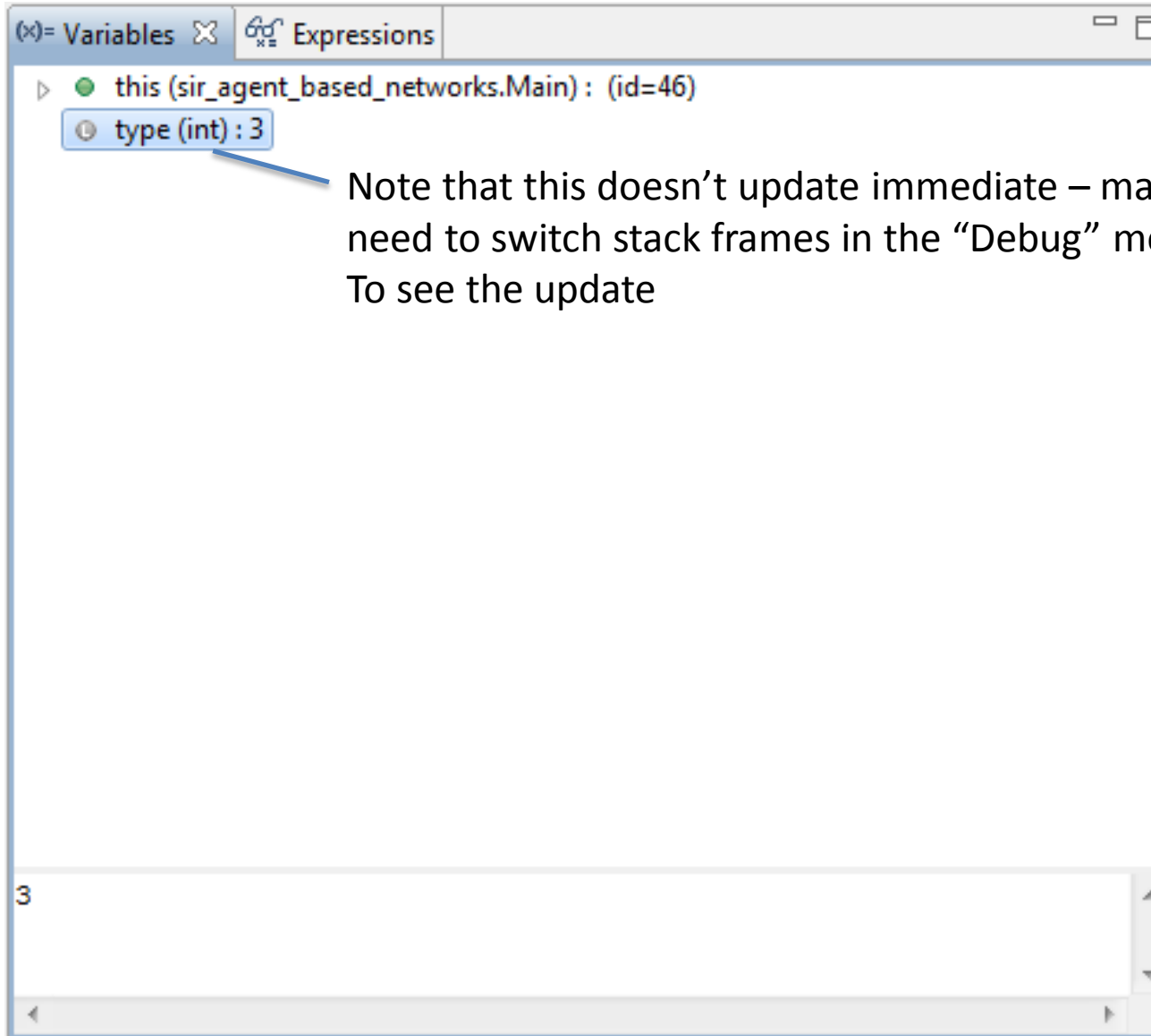
**Code Snippet:**

```
case text12: return "Infectivity:";
case text13: return
format( AverageIllnessDuration ) + " days"
;
case text14: return "Average illness duration:";
case text15: return
networkTypeToString( environment.getNetworkType() )
;
case text16: return "Network type:";
case text17: return
environment.getConnectionsPerAgent()
;
case text18: return "Links per agent:";
case text19: return "Maximum link distance: ";
case text20: return
environment.getConnectionRange()
;
case text21: return "Percent of long distance links:";
```

**Properties for `networkTypeToString - Function`:**

- Name:** `networkTypeToString`  Show name  Ignore  Public  Show at runtime
- Access:** `default`  Static
- Return type:**  void  boolean  int  double  String  Other: `String`
- Function arguments:**

# Seeing Result of Expression Evaluation



The screenshot shows a debugger window with two tabs: "Variables" and "Expressions". The "Expressions" tab is active and displays a tree view of the current stack frame. The root node is "this (sir\_agent\_based\_networks.Main) : (id=46)", which is expanded to show a child node "type (int) : 3". A blue callout box points to the "type (int) : 3" node with the text: "Note that this doesn't update immediate – may need to switch stack frames in the 'Debug' method To see the update". At the bottom of the window, a text box contains the value "3".

Note that this doesn't update immediate – may need to switch stack frames in the "Debug" method To see the update